VIDA
User Manual

version 3.0

OpenEye Scientific Software, Inc.

August 10, 2007

9 Bisbee Ct, Suite D
Santa Fe, NM 87508
www.eyesopen.com
support@eyesopen.com

Copyright © 1997-2007 OpenEye Scientific Software, Santa Fe, New Mexico. All rights reserved.

All rights reserved. This material contains proprietary information of OpenEye Scientific Software. Use of copyright notice is precautionary only and does not imply publication or disclosure.

The information supplied in this document is believed to be true but no liability is assumed for its use or the infringement of the rights of others resulting from its use. Information in this document is subject to change without notice and does not represent a commitment on the part of OpenEye Scientific Software.

This package is sold/licensed/distributed subject to the condition that it shall not, by way of trade or otherwise, be lent, re-sold, hired out or otherwise circulated without OpenEye Scientific Software's prior consent, in any form of packaging or cover other than that in which it was produced. No part of this manual or accompanying documentation, may be reproduced, stored in a retrieval system on optical or magnetic disk, tape, CD, DVD or other medium, or transmitted in any form or by any means, electronic, mechanical, photocopying recording or otherwise for any purpose other than for the purchaser's personal use without a legal agreement or other written permission granted by OpenEye.

This product should not be used in the planning, construction, maintenance, operation or use of any nuclear facility nor the flight, navigation or communication of aircraft or ground support equipment. OpenEye Scientific software, shall not be liable, in whole or in part, for any claims arising from such use, including death, bankruptcy or outbreak of war.

Windows is a registered trademark of Microsoft Corporation. Apple and Macintosh are registered trademarks of Apple Computer, Inc. AIX and IBM are registered trademarks of International Business Machines Corporation. UNIX is a registered trademark of the Open Group. RedHat is a registered trademark of RedHat, Inc. Linux is a registered trademark of Linus Torvalds. Alpha is a trademark of Digital Equipment Corporation. SPARC is a registered trademark of SPARC International Inc.

SYBYL is a registered trademark of TRIPOS, Inc. MDL is a registered trademark and ISIS is a trademark of MDL Information Systems, Inc. SMILES, SMARTS, and SMIRKS may be trademarks of Daylight Chemical Information Systems. Macromodel is a trademark of Schrödinger, Inc. Schrödinger, Inc may be a wholly owned subsidiary of the Columbia University, New York.

Python is a trademark of the Python Software Foundation.

Java is a trademark or registered trademark of Sun Microsystems, Inc. in the U.S. or other countries.

"The forefront of chemoinformatics" is a trademark of Daylight Chemical Information Systems, Inc.

Other products and software packages referenced in this document are trademarks and registered trademarks of their respective vendors or manufacturers.

CONTENTS

1 Introduction **1**
- 1.1 History . . . 1
- 1.2 OpenEye Toolkits . . . 2

2 Installation **3**
- 2.1 Microsoft Windows . . . 4
- 2.2 UNIX/Linux . . . 4
- 2.3 Mac OS X . . . 4
- 2.4 Licenses . . . 5

3 Getting Started **6**
- 3.1 General Concepts . . . 6
- 3.2 Scope . . . 7
- 3.3 Layout . . . 8
- 3.4 Undo/Redo . . . 11

4 File I/O **13**
- 4.1 Opening Files . . . 13
- 4.2 Closing Files . . . 16
- 4.3 Saving Files . . . 16
- 4.4 Importing Files . . . 19
- 4.5 Exporting Files . . . 20
- 4.6 Drag and Drop . . . 21

5 3D Display **23**
- 5.1 User Interaction . . . 23
- 5.2 Rendering . . . 32
- 5.3 Stereo . . . 32
- 5.4 Viewpoint . . . 33
- 5.5 Bookmarks . . . 34
- 5.6 Tiled Display . . . 34
- 5.7 Toolbars . . . 35
- 5.8 Display Widgets . . . 36

5.9 Molecular Visualization . . . 38
5.10 Grid Visualization . . . 46
5.11 Surface Visualization . . . 48
5.12 Custom Views . . . 54

6 2D Display **57**
6.1 Preview vs. Viewer . . . 58
6.2 Interaction . . . 59

7 Spreadsheet **60**
7.1 Basic Usage . . . 60
7.2 Sorting . . . 62
7.3 Displaying Data . . . 63
7.4 Organizing Columns . . . 63
7.5 Creating New Columns . . . 64
7.6 Filtering . . . 66
7.7 Statistics . . . 66
7.8 Formatting Columns . . . 68
7.9 Importing/Exporting . . . 69

8 List Management **72**
8.1 Introduction . . . 72
8.2 Browsing . . . 72
8.3 Views . . . 73
8.4 List Columns . . . 73
8.5 List Manipulation . . . 75
8.6 Toolbar . . . 77

9 Process Management **78**
9.1 Processes . . . 79
9.2 Pipes . . . 82

10 Scripting **83**
10.1 Scripting Window . . . 83
10.2 OEChemLite . . . 84

11 Preferences **87**

12 Examples **89**
12.1 Data . . . 89
12.2 Scripts . . . 90

13 Help **91**

A Menus **93**
A.1 File . . . 93
A.2 Edit . . . 96

A.3 View . . . 99
A.4 Data . . . 99
A.5 Style . . . 100
A.6 Bookmarks . . . 110
A.7 Tools . . . 111
A.8 Window . . . 111
A.9 Help . . . 113

B Mouse Maps **115**
B.1 VIDA Mouse Map . . . 115

C Release Notes **116**
C.1 Version 3.0.0 . . . 116
C.2 Version 2.1.2 . . . 117
C.3 Version 2.1.1 . . . 119
C.4 Version 2.1.0 . . . 120
C.5 Version 2.0.2 . . . 122

Bibliography **124**

CHAPTER

ONE

Introduction

VIDA is OpenEye's main visualization application designed to intuitively and effortlessly handle very large data sets while still generating extremely high quality images. VIDA provides multiple modes of display including 1D, 2D, and 3D displays. Furthermore, VIDA is an excellent interface for data analysis as it contains a chemically-aware spreadsheet and a powerful list-based architecture.

1.1 History

VIDA was originally developed to fill a very specific need – the ability to easily browse through a large number (>100,000) of small molecules. This was particularly important to OpenEye as many of our tools were capable of producing an enormous amount of data. At that time, there really was no acceptable solution to handle this amount of data in a user-friendly manner. Therefore, VIDA was created to solve this problem.

VIDA's intuitive and easy to use interface dramatically changed the way researchers were able to work and was widely adopted. Unfortunately, as the user-base continued to grow, so did VIDA. An enormous number of features were requested and in the end too many of them were implemented. Unfortunately, this significantly increased the bulk and decreased the stability of VIDA.

Another problem that became evident was VIDA's dependence on the *OELib* toolkit for chemistry handling. While *OELib* was an excellent toolkit for its time, it had an unfortunate "feature" in that it tried to be *smarter than you* and would attempt to correct the chemistry of the input molecules. Unfortunately, this often resulted in mangled chemistries much to the dismay of the user.

Fortunately, while VIDA was being developed, OpenEye embarked on an extensive and successful development project to create a new more robust chemistry toolkit - *OEChem*. With the completion and wide adoption of *OEChem* throughout OpenEye and our customers, it became clear that the time was right for the development of a new version of VIDA based on *OEChem*.

The decision to port VIDA to use *OEChem* prompted a lengthy discussion about the current design of VIDA. Extensive thought went into what should and more importantly what should not be a part of VIDA 2.0. The design goals were to reproduce the excellent visualization and browsing functionality of the first version, while emphasizing stability and simplicity of interaction. We believe we achieved these goals with the release of VIDA 2.0.

After successful tackling stability, we decided to focus our efforts on improving the user experience by making a cleaner and more simplified user interface. This triggered a significant redesign of the user interface which we hope will make VIDA easier to use and more accessible to all users. Given this large change, plus a large number of additional new features, we are releasing this newest version as VIDA 3.0.

We look forward to your comments and feedback.

1.2 OpenEye Toolkits

VIDA 3.0 was built using the following OpenEye toolkits:

- OEChem 1.5
- Ogham 1.5
- Lexichem 1.6
- Spicoli 1.0
- Zap 2.0

CHAPTER

TWO

Installation

This chapter provides the necessary details required to install VIDA 3.0. Currently supported platforms include:

- Microsoft Windows 2000
- Microsoft Windows XP
- Mac OS X 10.3 (PowerPC or Intel)
- Mac OS X 10.4 (PowerPC or Intel)
- RedHat Enterprise Linux 3.0
- RedHat Enterprise Linux 3.0 64-bit
- RedHat Enterprise Linux 4.0
- RedHat Enterprise Linux 4.0 64-bit
- RedHat Linux 9.0
- SuSE Linux 9.1
- SuSE Linux 9.1 64-bit
- SuSE Linux 10
- SuSE Linux 10 64-bit
- SGI Irix 6.5

Support for additional platforms is possible and will be considered on a case-by-case basis. To request support for additional platforms, please contact support@eyesopen.com.

Prior versions of VIDA do not need to be explicitly uninstalled in order to install the current version. The current version will, by default, install into a new directory specified by the version number. Multiple versions can co-exist on the same machine without difficulty.

2.1 Microsoft Windows

A simple installation wizard is provided for Microsoft Windows. Double-clicking on the file (e.g. `vida-3.0.0-microsoft-win32-i686-setup.exe`) will start the wizard which will direct the installation process. It is recommended that the installer to be allowed to install VIDA 3.0 in the `C:\OpenEye` directory if possible. Once the installation process is complete, there will be a VIDA icon on the desktop as well as in the start menu.

To uninstall VIDA, simply click on the *Uninstall VIDA* icon in the `C:\OpenEye` directory or in the start menu in the OpenEye/VIDA group. VIDA can also be uninstalled using the *Add/Remove Programs* option in the *Control Panel*.

2.2 UNIX/Linux

The UNIX/Linux installation is provided as a gzipped tar distribution (e.g. `vida-3.0.0-redhat-9.0-i586.tar.gz`). The first step is to determine where VIDA will be installed. We recommend `/usr/local`. Next move the installation file to that directory and untar the file using the following commands:

```
tar -zxvf vida-3.0.0-redhat-9.0-i586.tar.gz
```

This will create an `openeye` directory where VIDA will be installed. The executable can be found in the `openeye/bin` directory. It is worth considering adding `/usr/local/openeye/bin` (or wherever it is installed) to the `PATH` environment variable so that it can be executed from anywhere without having to specify the full installation path.

To uninstall VIDA, simply remove the executable from the `openeye/bin` direcctory. Nothing else is required. New versions can be installed directly over previous versions.

2.3 Mac OS X

The Mac OS X version comes pre-bundled. All that is required is to drag the VIDA icon to the desired location on the computer such as the Applications folder. Double-clicking on the icon will launch VIDA. The first time VIDA is run, VIDA may prompt for the location of the license file. If this occurs, there will be two VIDA icons in the Dock until VIDA is restarted.

To uninstall VIDA, simply drag the VIDA icon to the trash from the folder in which it was installed. Nothing else is required.

2.3.1 Platform differences

Microsoft Windows and Linux frequently use the *Ctrl* key as a part of a keyboard shortcut. However, on Mac OS X, the *Ctrl* key is often replaced by the *Apple* key for common operations such as copy, paste, and undo.

Similarly, in the 3D display, the *Alt* key must be held down to display popup information on Mac OS X instead of the *Ctrl* key as used on other platforms.

2.3.2 Launching from the command line

In a terminal window, VIDA can be launched from the command line as follows:

```
open -a vida
```

as long as VIDA is in the search path. If VIDA is installed into the Applications directory, it will automatically be in the search path.

2.4 Licenses

In order to run VIDA, a valid license provided by OpenEye must be present on the installed computer. OpenEye license files are typically named `oe_license.txt` and most OpenEye software products (including VIDA) expect to find that license file in the location specified by the `OE_LICENSE` environment variable or in the directory specified by the `OE_DIR` environment variable.

If neither of the above environment variables are defined on the computer, VIDA will prompt the user at run time to locate a valid license file. If a valid license file is specified, VIDA will remember the location of that file and will not prompt again until that license expires or is removed.

CHAPTER

THREE

Getting Started

This chapter is designed to help familiarize the user with the VIDA application and interface in general. While VIDA looks very similar on all platforms, there are some subtle differences between platforms, such as how each platform displays certain specific dialogs (the color selection dialog for instance). All screenshots provided in this manual were taken using the Microsoft Windows XP version of VIDA and therefore the actual appearance on other platforms may vary slightly from the images seen here.

3.1 General Concepts

There are a number of general concepts that are important to discuss early in order to explain how user interaction with VIDA works. VIDA is a stateful application and employs many special states (or properties) in its interaction with the user. The most important of these are: *Focused*, *Visible*, *Locked*, *Marked*, and *Selected*.

3.1.1 Focused State

The *Focused* state is a property of a single object in the application; there can be only one *Focused* object at any given time. Being *Focused* indicates that that particular object is the current object of interest and is therefore the primary focus of all relevant windows. If the *Focused* object is not already *Visible* (see Section 3.1.2), it will behave as if it were *Visible* for as long as it remains *Focused*.

In addition, when a molecule is the *Focused* object, its SMILES representation is displayed in the application's status bar (which can be found at the bottom of the application window). If any of the molecule's atoms are selected, that selection will be indicated in the SMILES display by the coloring of relevant atoms. Selected atoms are colored using the current *Selection Color* (the default is orange) and are also in a bolded font while all of the unselected atoms will be displayed in their usual normal black font.

For large molecules ($>$100 atoms in this case), no SMILES representation is displayed as it would not fit in the status bar.

3.1.2 Visible State

The *Visible* state is an indicator of whether or not the associated object is drawn in the 3D display (see Chapter 5). Multiple objects can be *Visible* simultaneously. If VIDA is operating in tiled mode, each *Visible* object will be displayed in its own individual pane.

If the 2D display (see Chapter 6) is also shown and VIDA is in tiled mode, *Visible* molecules will be drawn in the 2D display as well as the 3D display.

3.1.3 Locked State

The *Locked* state is essentially the same as the *Visible* state (see Section 3.1.2) except in the situation where the application is in tiled mode, in which case each *Locked* object will be displayed in every pane of the 3D display. However, a *Locked* object can also be simultaneously *Visible* and/or *Focused* (see section 3.1.1) in which case it will be displayed in its own individual pane as well as every other pane in the 3D display.

3.1.4 Marked State

The *Marked* state is simply an indicator of user interest in the associated object. The *Marked* state can be used to help filter data as well as to specify the desired input to application functions. For instance, the *Marked* state can be used to specify which molecules out of a large list are saved. Finally, when certain operations generate too many results to be displayed at one time, the *Marked* state can be used to indicate the results of those operations.

3.1.5 Selected State

The *Selected* state is a special property of the *Visible* state (see Section 3.1.2). Only objects that are *Visible*, *Focused* (see Section 3.1.1), or alternatively *Locked* (see section 3.1.3) can be *Selected*. Much like the *Marked* state (see Section 3.1.4), the *Selected* state is a indicator of interest and provides an input set to application functions. Unlike, the *Marked* state, the *Selected* state is more transient and is easily cleared. The act of *selection* is usually performed in either the 3D (see Chapter 5) or 2D (see Chapter 6) display. Selected objects are usually indicated by a change of color (the default is orange) in these displays. More details on the actual process of selection can be found in the 3D and 2D display chapters.

3.2 Scope

Scope is not a specific state or property like those discussed in the previous section but is rather an indicator as to which objects the application should operate on. The default application scope is *Focused* which means that all of the application functions will operate on the *Focused* object (see Section 3.1.1).

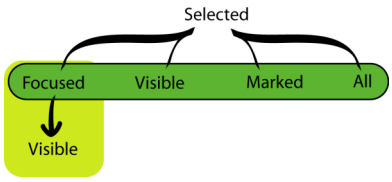


Figure 3.1: Available Scopes

However, as seen in figure 3.1, if there is anything currently *Selected* (see Section 3.1.5), the selected set will take precedence over the current scope. Furthermore, in *Focused* scope, if there is no *Selected* set and no *Focused* object, the *Focused* scope defaults to behaving as if it were the *Visible* scope.

The *Visible* scope operates on all objects that are currently *Visible* (see Section 3.1.2), *Locked* (see Section 3.1.3), and/or *Focused*.

The *Marked* scope operates on all objects that are currently *Marked* (see Section 3.1.4).

The *All* scope operates on every object currently loaded in the application. Operating on the *All* scope can be a very lengthy process, and as such, its progress is displayed in the progress bar in lower right hand corner of the application. Fortunately, operations in the *All* scope can be halted if desired by clicking on the *Stop* button immediately to the left of the progress bar.

The current scope is displayed in and can be modified from the main application toolbar.

3.3 Layout

The layout of the application should be relatively familiar to most users. VIDA provides a menu bar with many standard as well as specific menus, a toolbar for common operations, a central main window, and a number of peripheral windows. An example of the layout can be see in Figure 3.2.

The layout including the position and visibility of the main and peripheral windows is saved on exit and will be restored the next time the application is run. The layout of the windows can be changed by clicking on the title bar of the individual window and dragging it to the desired location. A placeholder window will appear when the mouse is over an area where the dragged window can be placed as seen in Figure 3.3.

Furthermore, multiple windows can be tiled on top of each other into a tab controlled area as can be seen in the bottom window in both Figures 3.2 and 3.3.

In addition to manually changing the layout, there are a number of predefined layouts which are accessible via a drop down button on the toolbar as can be seen in Figure 3.4. The colored icon associated with each option indicates which peripheral windows will be shown and where. Hovering the mouse over any given option for about one second will generate a preview image of the resulting layout. The

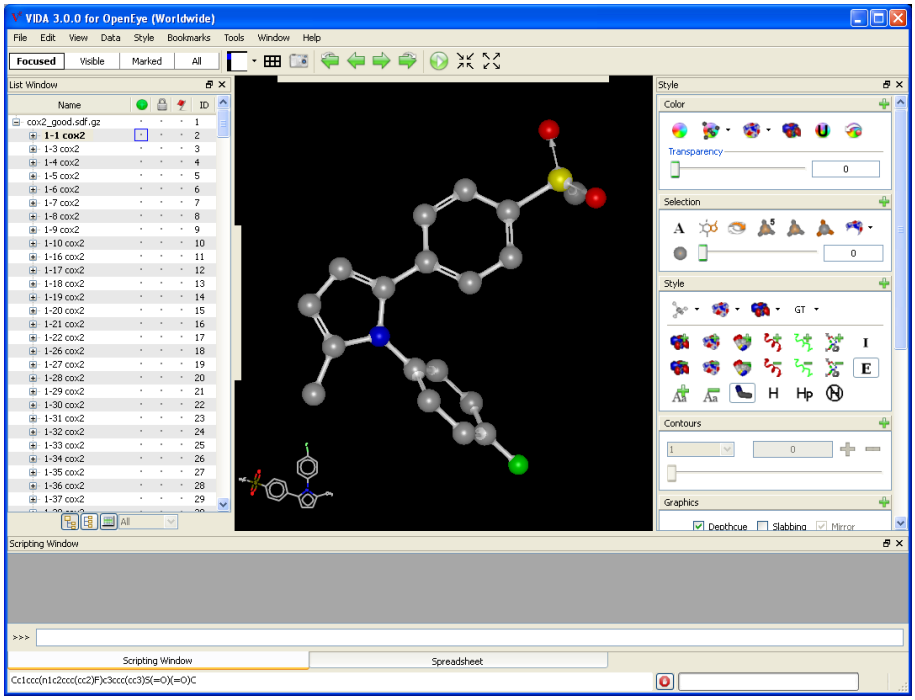


Figure 3.2: Docked window layout

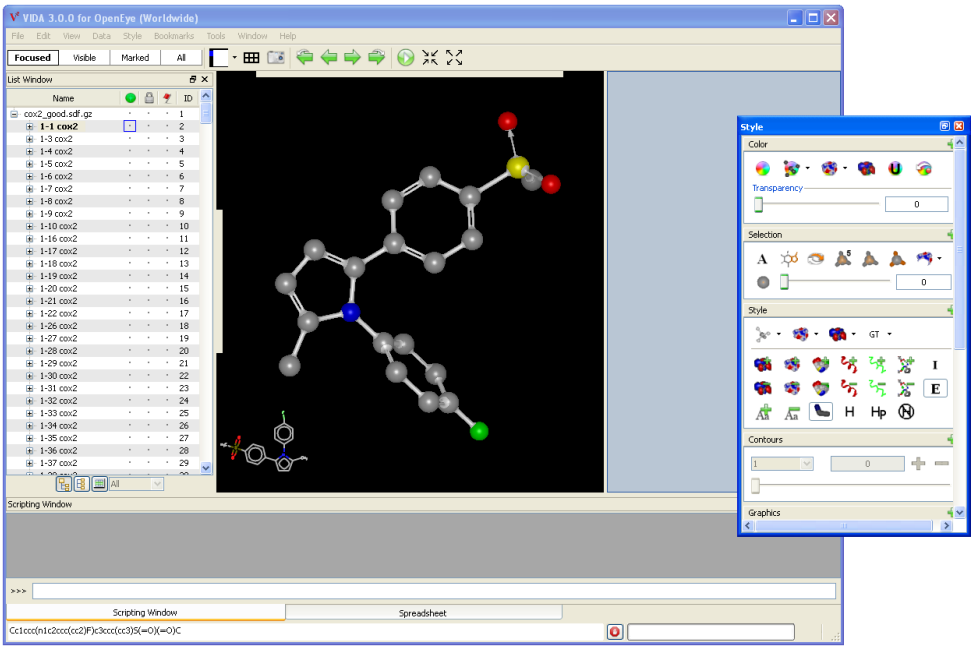


Figure 3.3: Changing the layout by dragging

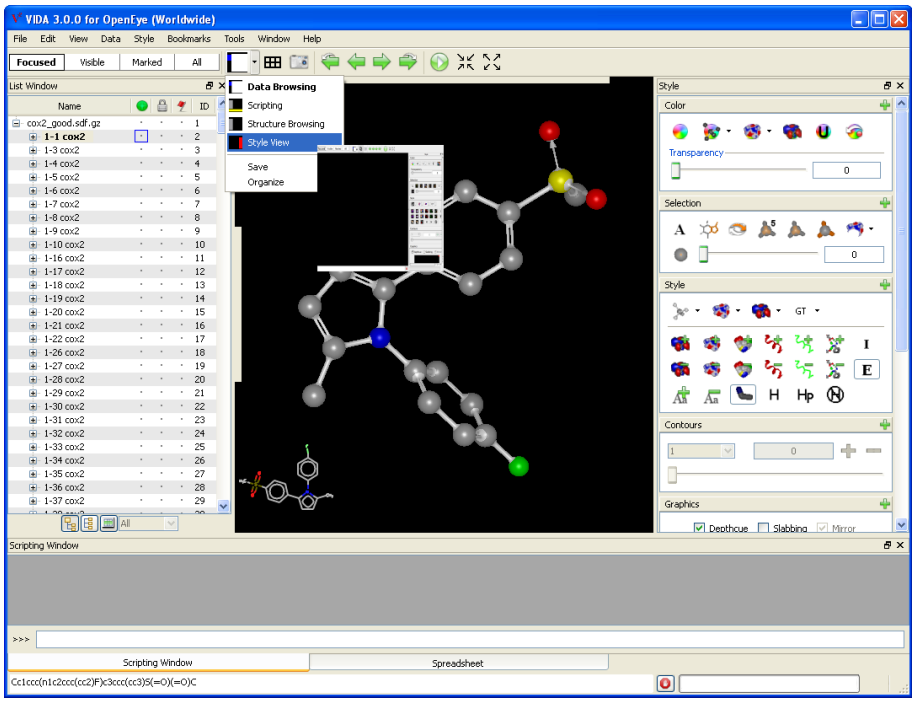


Figure 3.4: Changing the layout from the toolbar

"Save" option allows for the creation and storage of custom layouts. The "Organize" option enables the reorganization of the ordering of the layouts.

3.3.1 Main Windows

The main window is the window that occupies the central area of the application on the screen. At any given time, there can be only one main window; however, there are multiple windows which are capable of being the main window if desired. The choice of main window can be controlled by selecting the desired option from the "Main window" submenu in the top-level *Windows* menu. In Figure 3.2, the 3D display (see Chapter 5) is serving as the main window.

Currently, there are three different main window options: the 3D display, the 2D display (see Chapter 6), and the spreadsheet (see Chapter 7). More details about the individual windows can be found in their relevant chapters.

Fullscreen

In addition to the standard layout described above, the main window can optionally be displayed in fullscreen mode, by selecting the "Fullscreen" option in the top-level *View* menu. Pressing the "Esc" key will exit fullscreen mode (please note that clicking the mouse on the screen may be required before hitting the "Esc" key in order to ensure that the key press is registered).

When in fullscreen mode, the main menu bar and status bar are hidden by default. However, they can be toggled on or off by hitting the “F1” and “F2” keys respectively.

The 3D display normally contains popup toolbars around the edge of the window. These toolbars remain when in fullscreen mode to allow for easy access to display functionality. For more details on these toolbars see Chapter 5.

3.3.2 Peripheral Windows

In addition to the main window, there can be multiple peripheral windows docked around the edges of the main window or as separate floating top level windows. Furthermore, multiple windows can be placed on top of each other in a tabbed region.

Currently, multiple peripheral windows (including the potential main windows) are available:

- 2D Display - see Chapter 6
- 2D Preview - see Chapter 6
- 3D Viewer - see Chapter 5
- List Window - see Chapter 8
- Process Manager - see Chapter 9
- Scripting Window - see Chapter 10
- Spreadsheet - see Chapter 7
- Style Control - she Chapter 5

The display of peripheral windows is controlled by toggling the desired window options in the *Windows* menu. Currently visible windows are indicated by the presence of a check mark next to their names. Peripheral can also be hidden by clicking on the ‘X’ in the upper right hand corner of the window. As mentioned above, the layout of the peripheral windows can be controlled by dragging the individual windows to their desired locations (see Figure 3.3).

3.4 Undo/Redo

VIDA provides multi-step undo and redo functionality of almost all operations. Operations can be undone by selecting the “Undo” option in the *Edit* menu or by hitting *Ctrl+Z* on the keyboard. Either of these will undo the last operation performed. This can be done repeatedly or the undo history can be viewed in the “Undo History” option the *Edit* menu which allows selecting how far back into the undo history to go. Please note that while this menu only displays the 10 most recent operations, the undo history can be much greater, so feel free to revisit the “Undo History” option to go back further.

Operations that were just undone can be redone by selecting the "Redo" option in *Edit* menu or by hitting *Ctrl+Y* on the keyboard. Either of these will redo the last operation undone. This can be done repeatedly or the redo history can be viewed in the "Redo History" option in the *Edit* menu which allows selecting how far into the redo history to go. Like the undo history, this menu only displays the 10 most recent undone operations, but the redo history may be much greater and can be revisited to go further.

CHAPTER

FOUR

File I/O

4.1 Opening Files

There are multiple mechanisms by which a file can be loaded in VIDA. The simplest method is to select the "Open" option in the *File* menu. This will launch a file selection dialog which prompts the user for the desired files(s) which will subsequently be loaded. A short list of the most recently opened files can also be found in the *File* menu under the "Recents" (or "Open Recents" on Mac OS X) submenu. Selecting an option in this submenu will load the associated file. There is an "Open All" option at the bottom of the recents menu which will load all of the files listed in the recents submenu. Files can also be loaded by specifying them on the command line when starting VIDA or by dragging the file of interest onto the VIDA desktop icon. By default, there can only be one instance of VIDA running at a time, so if an instance of VIDA is running and a new file is specified on the command line or if a file is dragged onto the desktop icon, that file will be loaded in the currently running instance (instead of loading a new instance). If desired, multiple instances can be enabled by selecting the "Allow multiple instances" option in the application preferences (see Chapter 11).

In addition to the "Open" menu item, there is a separate "Open State" item which filters out all files except VIDA state files. State files can be opened using any of the mechanisms described above, but their special nature warranted their own menu item. For more details on state files see Section 4.1.5.

There is also an "Open Special" submenu which contains a "Script" option which like the "Open State" option filters out all files except for Python scripts. The "Open Special" submenu also serves as an excellent target area for adding custom open operations via the scripting interface (see Chapter 10).

4.1.1 Molecules

Multiple molecular file formats are supported for reading and include:

- OpenEye binary format v1 and v2 (.bin and .oeb)
- MDL, RDF, and SDF

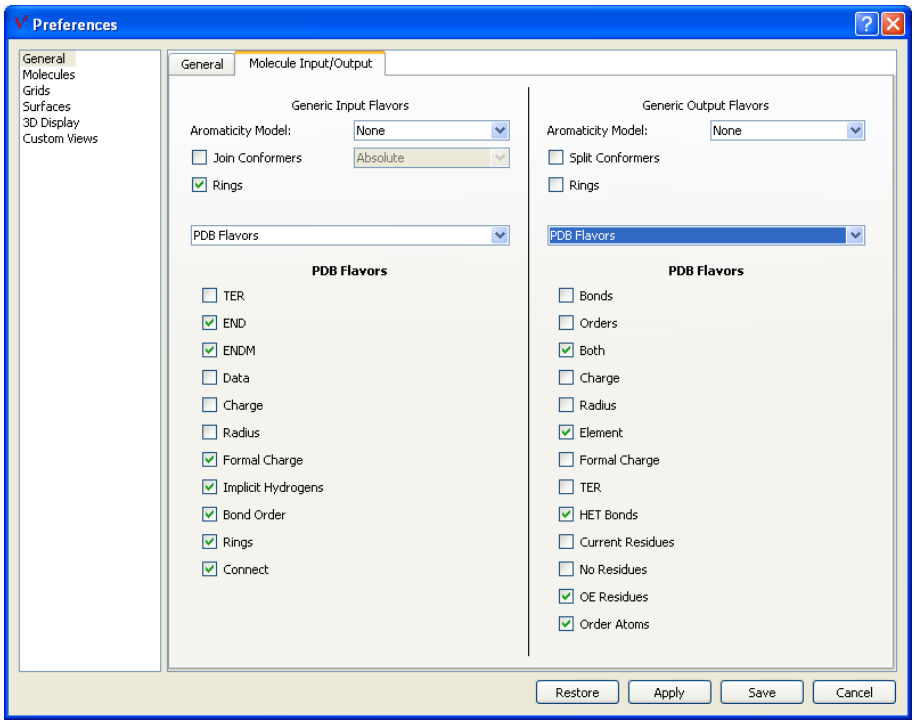


Figure 4.1: Advanced molecule input and output options

- Tripos MOL2, MOL2H
- Daylight SMILES, canonical SMILES, and isomeric SMILES
- ChemDraw CDX
- ISIS Sketch
- MacroModel
- MOPAC
- PDB
- XYZ

As is often the case with file formats, the meaning and use of certain fields within a format may change over time which can potentially lead to problems interpreting those files. VIDA, using OpenEye's OEChem toolkit, makes its best effort to interpret all files correctly; however, it does provide a mechanism to allow the user to override the handling of certain formats (SMILES, PDB, Mol2, XYZ, and MacroModel). The ability to change the "flavor" of a specific format can be done in the application preferences as seen in Figure 4.1.

In addition to changing the flavor of a format, there are a few other advanced options available when reading molecules including aromaticity model specification and conformer joining. The desired aromaticity model to be applied can be specified in the pulldown menu next to the "Aromaticity Model"

label. The “Join Conformers” checkbox controls whether or not adjacent molecules in an input file will be tested on reading to determine whether they are unique compounds or simply different confomers of the same molecule. The specific test to be performed can be specified in the pulldown menu next to the checkbox. For more specific details on the available aromaticity models and conformer tests, please see the OEChem documentation.

It is important to note that because these advanced options are available through the application preferences, they will be remembered by the application and automatically applied in future “Open” operations unless they are subsequently restored.

4.1.2 Grids

Multiple grid and map formats are supported for reading and include:

- OpenEye grids
- ASCII grids
- GRASP/Delphi grids
- CCP4 maps
- XPLOR maps

4.1.3 Surfaces

Multiple surface formats are supported for reading and include:

- OpenEye surfaces
- GRASP surfaces

4.1.4 Python Scripts

VIDA supports reading of Python scripts compatible with Python version 2.3. VIDA supports two additional file extensions for Python scripts (.pyv and .vpy) to enable users to associate those file extensions with VIDA without adversely affecting the association of unrelated Python scripts.

For more details about Python scripting, please see Chapter 10 as well as the associated Python scripting API documentation.

4.1.5 State

The entire state of a session can be stored in a single file called a "State File" (.oes). A state file contains all of the molecules, grids, and surfaces that were loaded in VIDA at the time the state file was created. In addition, the state file preserves the actual view, layout, and properties of that session. State files provide an extremely convenient way to save sessions for later work or to share with collaborators. Furthermore, state files are the fastest method of reading and writing large data sets.

To load a state file, choose the "Open State" option in the *File* menu and select the desired state file. State files can also be loaded from the commandline. It is important to note that loading a state file during a run will clear the current state before loading the new one. Therefore, be sure to save the current state before loading a new state if keeping the current state is desired. Furthermore, since state files contain the entire state of the application that generated them, loading a state file will overwrite local preferences; however, they will not be saved on exit.

4.2 Closing Files

When a file is opened in VIDA, its contents are loaded into its own individual list. To close a given file, simply right-click on the associated list in the List Window (see Chapter 8) and select the "Delete" option. This will close the file and remove the contents from the application. However, if an object is present in multiple lists, deleting one of those lists will not delete that object from the other lists. It is important to note that deleting a list, does not affect the actual file that was read.

To close all of the currently loaded files, select the "Clear All" option in the *File* menu. In addition to clearing everything that is loaded, this option effectively restores the state of the application to what it was in when the application was started.

4.3 Saving Files

VIDA is capable of saving multiple types of files including molecules, grids, surfaces, and state files. The details of the individual formats can be found in the relevant sections below.

A file can be saved by choosing the "Save" option the *File* menu. This will launch a dialog prompting the user to select which objects to save (see Figure 4.2). There are two sets of buttons at the top of this dialog which provide mechanisms for filtering the available options. The three buttons on the top left toggle the display of molecules, surfaces, and grids respectively. The two buttons on the top right provides filters for showing only the *Marked* and/or *Visible* objects respectively. Once one or more objects have been selected and the "OK" button is pressed, a file dialog will appear allowing the user to specify the desired file. The file formats listed in the file dialog filter will reflect which file formats are supported based on the objects selected. Only the OpenEye binary format supports writing multiple object types (molecules, grids, and/or surfaces) to the same file.

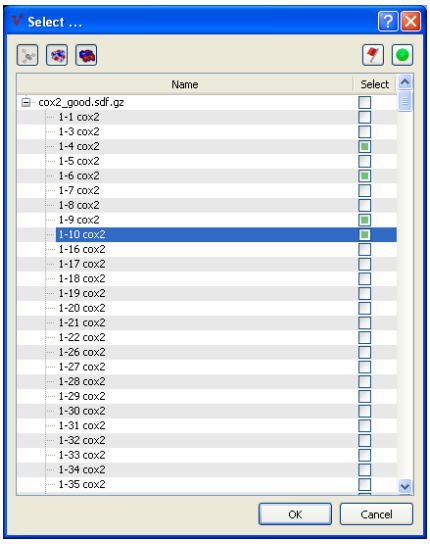


Figure 4.2: Selecting what to save

This process can be expedited by simplying right-clicking on the desired objects and selecting the “Save” option. This allows the user to bypass the selection dialog and go directly to the file dialog.

4.3.1 Molecules

Multiple molecular file formats are supported for writing and include:

- OpenEye binary format v2 (.oeb)
- MDL, RDF, and SDF
- Tripos MOL2, MOL2H
- Daylight SMILES, canonical SMILES, and isomeric SMILES
- ChemDraw CDX
- MacroModel
- Molecular Formula (MF)
- MOPAC
- FASTA
- PDB
- XYZ

As is often the case with file formats, the meaning and use of certain fields within a format may change over time which can potentially lead to problems interpreting those files. VIDA, using OpenEye's OEChem toolkit, writes all files according to the standard; however, it does provide a mechanism to allow the user to override the handling of certain formats (SMILES, MDL/SDF, PDB, Mol2, MF, MOPAC, and MacroModel). The ability to change the "flavor" of a specific format can be done in the application preferences as seen in Figure 4.1.

In addition to changing the flavor of a format, there are a few other advanced options available when writing molecules including aromaticity model specification and conformer splitting. The desired aromaticity model to be applied can be specified in the pulldown menu next to the "Aromaticity Model" label. For more specific details on the available aromaticity models, please see the OEChem documentation.

It is important to note that because these advanced options are available through the application preferences, they will be remembed by the application and automatically applied in future "Save" operations unless they are subsequently restored.

4.3.2 Grids

Multiple grid formats are supported for writing and include:

- OpenEye grids
- ASCII grids
- GRASP/Delphi grids

4.3.3 Surfaces

Multiple surface formats are supported for writing and include:

- OpenEye surfaces
- GRASP surfaces

4.3.4 State

The entire state of a session can be stored in a single file called a "State File" (.oes). A state file contains all of the molecules, grids, and surfaces that were loaded in VIDA at the time the state file was created. In addition, the state file preserves the actual view, layout, and properties of that session. State files provide an extremely convenient way to save sessions for later work or to share with collaborators. Furthermore, state files are the fastest method of reading and writing large data sets.

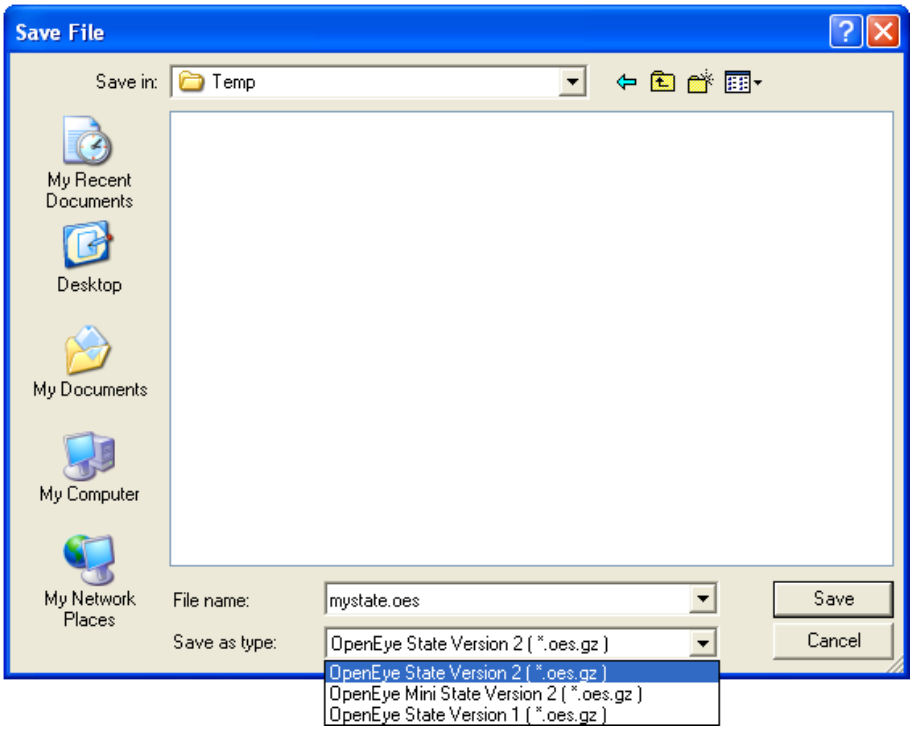


Figure 4.3: Saving state

With the release of VIDA 3.0, a new state file version was introduced which is significantly smaller in size and faster to load than previous versions. However, this new version, "OpenEye State Version 2" cannot be read or written by VIDA 2.x. To address that issue, VIDA does provide the functionality to save "OpenEye State Version 1" files which can be read and written by all versions of VIDA.

The new state version also supports a "mini" option which only writes out objects that are currently visible or referenced in the display bookmarks. This can provide for an even greater savings in disk size and load time. This mini-state file can be further compressed by selecting the "Compress grid contour information" option in the *General* section of the application preferences. Selecting this option zeros all non-contour related positions in the grid which greatly improves its compressibility. This should only be done if there is no expectation that the user loading the resulting state file will want to look at grid contour levels other than those already defined when the file was generated.

To save a state file, select the "Save State" option in the *File* menu and specify the desired state file. The state file version can be selected using the file format filter as seen in Figure 4.3.

4.4 Importing Files

External spreadsheet data can be imported into VIDA and associated with currently loaded objects. Both comma separated files (.csv) and tab delimited files (.txt) are supported by default. Once a file has been

selected, an import dialog will appear which allows the user to customize the column delimiters, column headers, as well as the criteria on which rows are matched to currently loaded objects. For a more specific details on importing spreadsheet data, please see Section 7.9.

4.5 Exporting Files

Multiple special file types can be exported from VIDA including spreadsheet data, images, and scripts. Exact details about these files can be found in the relevant sections below.

4.5.1 Data

Spreadsheet data can be exported in two formats: comma separated files (.csv) and tab delimited files (.txt). Data can be exported by selecting the "Spreadsheet" option in the *Export* submenu in the top-level *File* menu. Selecting this option will launch a file dialog which allows the user to specify the output file. Once the desired file has been chosen, another dialog will appear which allows the user to select which spreadsheet and which columns in that spreadsheet should be exported. For more specific details on exporting spreadsheet data please see Section 7.9.

4.5.2 Images

Screenshot

A screenshot of the current main window (see Section 3.3) can be exported in an variety of image formats to a user-specified file by selecting the "Screenshot" option in the *Export* submenu in the *File* menu. This action can also be achieved by clicking on the button with the camera icon in the application toolbar.

Selecting this option will launch a dialog showing a preview of the screenshot to be taken as seen in Figure 4.4. The user can also adjust the desired image resolution as well as specify whether or not to capture the image in black and white (which can be useful for publication purposes). The output file is specified at the bottom of the dialog.

POV-Ray

The scene in the 3D display can be exported to a POV-Ray input file which can be used to generate very high quality and resolution images. To export a POV-Ray file, select the "POV-Ray" option in the *Export* submenu in the *File* menu. The application will then prompt the user to specify a file to which the scene will be saved.

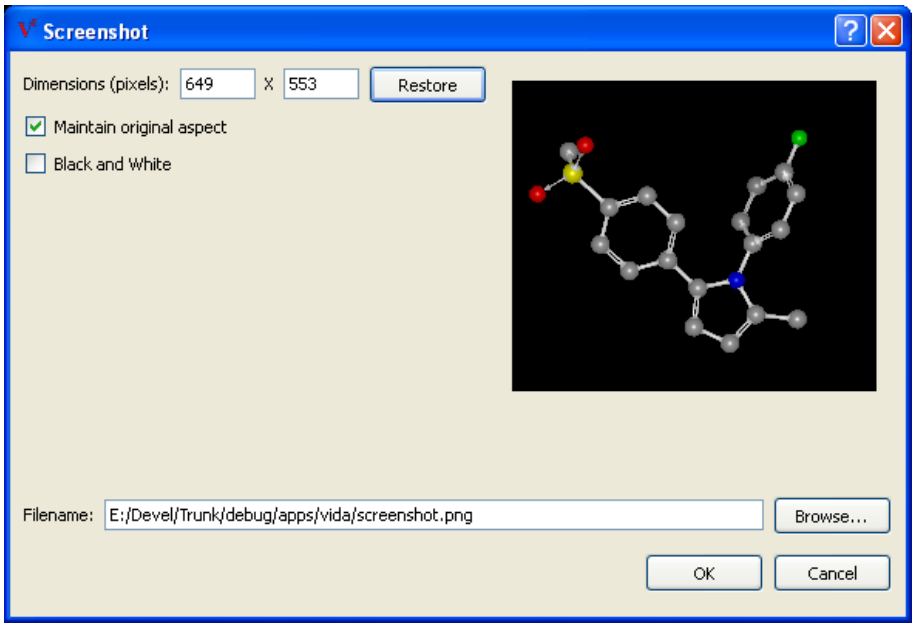


Figure 4.4: Capturing a screenshot

4.5.3 Python

A complete history of the application's operations can be written out as a Python script. This script can be exported by selecting the "Script history" option in the *Export* submenu in the *File* menu.

4.6 Drag and Drop

A powerful drag and drop interface is provided for passing molecules, images of molecules, and molecular data between applications. Multiple mechanisms of transfering data via the drag and drop interface are supported.

- Dragging a list of filenames into the List Window (see Chapter 8) will cause those files to be loaded.
- Dragging a list of SMILES into the List Window will cause those SMILES to be parsed into new molecules and stored in a new list called "Pasted". These new molecules will not be assigned 3D coordinates and therefore will not be visible in the 3D display (see Chapter 5).
- Dragging a list of IUPAC or common chemical names into the List Window will cause those names to be parsed into new molecules (using Lexichem) and stored in a new list called "Pasted". Please note, this functionality requires a separate license for the Lexichem toolkit available from OpenEye. These new molecules will not be assigned 3D coordinates and therefore will not be visible in the 3D display (see Chapter 5).

- Dragging molecules from other molecular visualization programs into the List Window will cause those molecules to be loaded into a new list called "Pasted". The ability to load molecules from other applications depends on how the other applications place molecules on the clipboard. Currently supported clipboard formats include SMILES, ChemDraw CDX, ISIS Sketch, MDL SD Files, and OpenEye OEB files. Furthermore, for other molecular visualization programs that support a drag and drop interface, molecules can be dragged from the List Window into those applications.
- Dragging molecules from the List Window into other applications will cause molecules, molecular data, or molecular images to be transfered to that application depending on how that application expects to receive data. For instance, dragging molecules into a Text Editor will result in a list of SMILES corresponding to the dragged molecules to be entered into that application.
- Dragging a 2D depiction from the Spreadsheet (see Chapter 7) will cause that image to be loaded in the associated application if possible.

Along the same vein, a similar copy and paste interface exists for passing molecules, images of molecules, and molecular data between applications. The "Copy" option can be found in the *Edit* menu and can also be performed by pressing *Ctrl+C* on the keyboard. This will copy all of the molecules in the current scope to the clipboard, unless the active window is the Spreadsheet (see Chapter 7) in which case it will copy all of the selected Spreadsheet data to the clipboard instead.

The "Paste" option can be found in the *Edit* menu and can also be performed by pressing *Ctrl+V* on the keyboard. This will paste any molecules found on the clipboard into a new list called "Pasted".

CHAPTER

FIVE

3D Display

The 3D display is the primary visualization interface available to the user (although it is not the only one). The purpose of the 3D display is to allow the user to view and interact with molecules (see Section 5.9), grids (see Section 5.10), and surfaces (see Section 5.11).

5.1 User Interaction

The primary interface mechanism to the 3D display is the mouse which is discussed in detail in Section 5.1.1. However, there are a large number of potential operations that can be performed in the 3D display which necessitates an additional mechanism of interaction besides the simple use of mouse buttons and popup menus. For this reason a separate peripheral window (see Section 3.3.2) is provided to access these operations. This window is called the "Style Control" and is discussed below in Section 5.1.2.

5.1.1 Mouse

As described above, the mouse is considered to be the primary interface mechanism to the 3D display. The details of how to use the mouse to interact with the 3D display are listed here. A three (or two) button mouse is recommended for use; however, support for a single button mouse is available. Holding down the *Ctrl* key while clicking or pressing the mouse button will emulate using the *Right* mouse button on a three button mouse.

For users not comfortable or familiar with the mouse interactions described below, VIDA can be made to emulate the mouse behavior of many other applications including: Coot, Insight, Maestro, MOE, O, Quanta, PyMol, RasMol, and Sybyl. The desired mouse map can be set in the application preferences in the *3D Display* section. Please note that not all the functionality described below is available in the emulated mouse modes, nor is all of the functionality of the emulated applications available in VIDA.

Selection

Objects in the display can be selected by clicking on them using the *Left* mouse button. The selection is cleared between button clicks (or by clicking in the background) unless the *Shift* or *Ctrl* key is also held down when clicking. If a single vertex on a surface has been selected, holding down the *Shift* key and selecting an second vertex will select the shortest path of vertices between the two selected end points.

Double-clicking using the *Left* mouse button expands the current selection to the next logical grouping. For instance, double-clicking on a selected atom in a protein will expand the selection to include all of the atoms in the same residue as the original selected atom. Double-clicking on the selected set again will expand the selection to include either the entire chain if there are multiple chains, otherwise it will include the entire molecule.

Selection can also be performed using a lasso style selection method by holding down the *Right* mouse button and moving the mouse in the window to define a selection rectangle (a dotted rectangle will be displayed on the screen as you do this). Releasing the *Right* mouse button will select all of the atoms and bonds inside of the rectangle. The previous selection will be cleared unless the *Shift* or *Ctrl* key is also held down while moving the mouse. Lasso style selection of surfaces and grids is not currently supported.

Additional methods of selection are available in the *Selection* pane of the Style Control (see Section 5.1.2).

Rotation

The scene can be rotated in three dimensions by holding down the *Left* mouse button and moving the mouse in the window. These movements will emulate rotation using a trackball.

The scene can also be rotated around just the Z axis by holding down the *Alt* key and the *Left* mouse button simultaneously.

Translation

Using the mouse, the scene can be translated in the XY plane by holding down the *Shift* key and the *Left* mouse button simultaneously. Translation along the Z axis can be performed by holding down the *Alt* key, *Shift* key, and the *Left* mouse button simultaneously. The scene can also be translated along the Z axis by holding down the *Alt* key while using the mouse wheel.

Using the keyboard, the scene can be translated horizontally in the display by pressing either the *A* key or *D* key which translate to the left and to the right respectively.

Scale/Zoom

The scene can be scaled (or zoomed) by holding down the *Middle* mouse button, the *Left* and *Right* mouse buttons together, or by using the mouse wheel. Any of these three modes will scale the scene. Multiple modes are provided for convenience and to accommodate the many varied mouse configurations in existence.

The scene can also be scaled using the *W* and *S* keys on the keyboard.

Text Scale

The scale of the text displayed in the scene can be controlled by using the mouse wheel while holding down the *Ctrl* key.

Clipping

The position of the near and far clipping (or slabbing) planes can be controlled using the mouse wheel while holding down both the *Ctrl* and *Shift* keys. Both planes are moved simultaneously and mirror each others' positions. It is important to note that slabbing must already be enabled for these operations to actually be performed.

Contour Level

The contour level of the grids in the default scope can be adjusted using the mouse wheel while simultaneously holding down the *Shift* key. Each incremental turn on the mouse wheel corresponds to an increase or decrease in the contour level by 0.1.

Label

Informative labels can be displayed about atoms and bonds (as well as surfaces and grid contours) underlying the current mouse position if the *Ctrl* key is pressed while moving the mouse (no mouse button need be depressed) on most platforms. On the Mac, this behavior is obtained by holding down the *Alt* key instead of the *Ctrl* key.

Additional methods of labeling atoms and bonds are available in the *Style* pane of the Style Control (see Section 5.1.2).

Menu

A context sensitive popup menu can be generated by clicking in the window while holding the *Right* mouse button down. This menu will contain operations that can be performed on those objects associated with the context when the popup menu was generated.

5.1.2 Style Control

The Style Control is a separate peripheral window (see Section 3.3.2) which is provided to enable additional interactions with the 3D display. The Style Control is a vertical window consisting of many individual panes. Each pane provides its own set of functionality. The display of the individual panes can be toggled by clicking on the title bar of the pane of interest. In addition, individual panes can be "torn off" to act as top-level windows by clicking on the "+" icon in the top right of the title bar. When

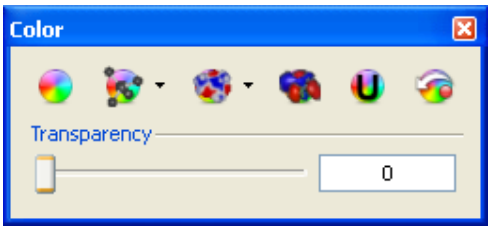


Figure 5.1: Color pane of the Style Control

the pane is torn off the "+" icon becomes a "-" icon which when clicked hides the "torn off" window. The specific details of the individual panes are listed below.

Color

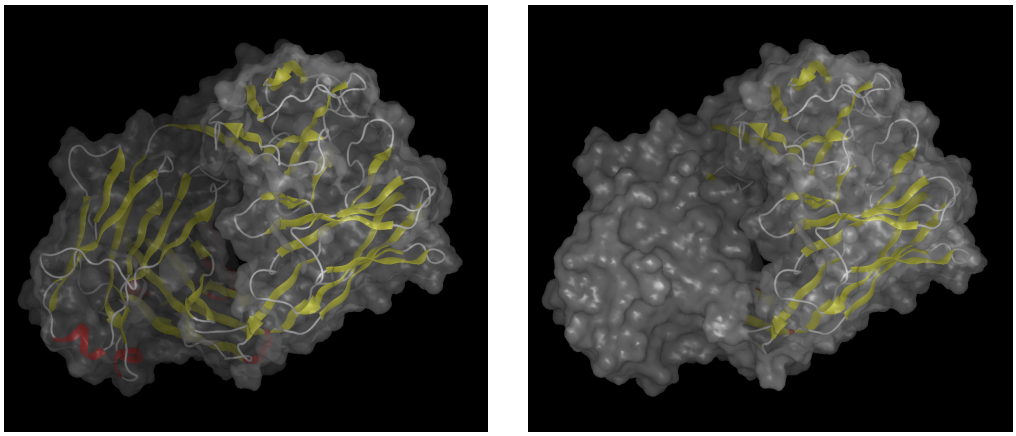
VIDA provides a wide array of coloring options as can be seen in the color pane (see Figure 5.1). The first button (with the color wheel icon) prompts the user to select a specific color which will be applied to all of the objects in the current scope. The second button (with the molecule overlayed on a color wheel) prompts the user to select a specific color that will be applied to all molecules in the current scope. In addition, this button contains a drop down menu of molecule specific coloring schemes which can be applied. Detailed information about the individual color schemes can be found in Section 5.9.2. The third button (with the surface overlayed on a color wheel) prompts the user to select a specific color that will be applied to all surfaces in the current scope. In addition, this button contains a drop down menu of surface specific coloring schemes which can also be applied. Detailed information about the individual color schemes can be found in Section 5.11.2. The fourth button (with the grid overlayed on a color wheel) prompts the user to select a color that will be applied to all grid contours in the current scope. The fifth button (with the "U" icon overlayed on a color wheel) assigns a unique color to every object in the current scope. The last button restores the original color of all the objects in the current scope. More details about the specific coloring schemes can be found below.

Beneath the buttons is a slider which can be used to adjust the transparency of surfaces (see Figure 5.2(a)) and grid contours. The slider values range from 0 (completely opaque) to 100 (completely transparent). The specified transparency value is applied to all surfaces and grid contours in the current scope. However, if a part of a surface is selected, the transparency will only be applied to the selected region as can be seen in Figure 5.2(b).

Selection

The simplest mechanism to perform a selection is to click on the object of interest in the 3D display using the mouse as described in Section 5.1.1. However, a number of addition selection mechanisms are available in the selection pane as can be seen in Figure 5.3.

In the top row of the selection pane, there are seven individual buttons. The first button with the "A" icon selects all of the currently visible molecules. The second button with the partially selected molecule icon performs a selection based on a substructure query. Clicking on this button will launch a query dialog which allows the user to specify a substructure query. The substructure can be entered as a SMARTS



(a) Transparent surface (b) Partially transparent surface

Figure 5.2: Transparent surfaces with underlying ribbon display visible

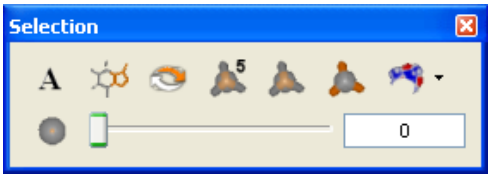


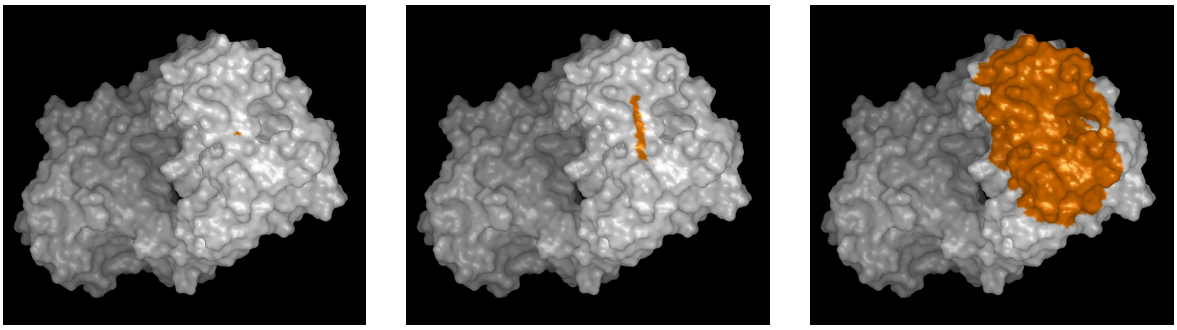
Figure 5.3: Selection pane of the Style Control

pattern, selected from a large number of predefined patterns, or can be specified using a common or IUPAC name which will be converted to a structure using OpenEye's Lexichem toolkit. All atoms in the current scope matching the specified pattern will be selected. If the current scope is set to *All* and more than 50 molecules match the pattern, the user will be prompted as to whether to *Mark* the matching structures instead of selecting them, as being selected would make them visible.

The third button inverts the current selection, unselecting everything that was selected and selecting everything that is currently visible but was not previously selected. The fourth button selects everything within a defined radius of the current selected set (the default is 5 Angstroms). The fifth button selects everything within a user specified radius of the current selected set. This radius is specified by the slider in the bottom row of the pane. The sixth button selects everything outside of the same user specified radius of the current selected set. It is important to note that these distance based selection mechanisms expand their selected sets to include entire residues in the event that only a portion of a residue is included within the radius.

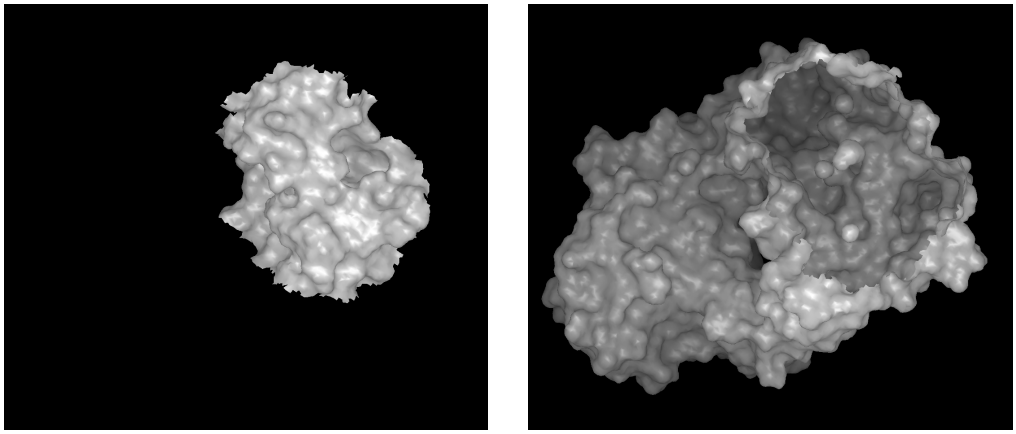
The last button contains a drop down menu which has different options based on the current selection. If a molecule is selected it contains the following options: "Hide Outside", "Hide Inside", and "Restore". Selecting one of these options will hide all of the atoms and bonds outside the current slider specified radius, hide all of the atoms and bonds inside the current slider specified radius, or unhide any previously hidden atoms and bonds respectively. If a surface is the options are: "Crop Outside", "Crop Inside", and "Restore". Selecting one of these options will crop away the unselected portion of the surface, crop away the selected portion of the surface, or restore the surface to its original state respectively. Examples of cropped surfaces can be seen in Figure 5.5.

In the bottom row, there is one button adjacent to a slider and a numeric display. The button is a toggle that controls whether or not unselected atoms and bonds are shown based on their distance to the selected set. The cutoff distance is controlled by the adjacent slider (and the actual value in Angstroms can be seen in the numeric display). This state of this button is ignored when a surface is selected. When a vertex (or a line of vertices) is selected, the slider can be used to flood out from the original selected set to include many more adjacent vertices. The different types of surface selections can be seen in Figure 5.4.



(a) Single vertex selected (b) Line of vertices selected (c) Region of vertices selected

Figure 5.4: Collection of different possible selection states on a surface



(a) Unselected region removed (b) Selected region removed

Figure 5.5: Different styles of cropping a surface

Style

The style pane of the Style Control is divided into two distinct areas as can be seen in Figure 5.6. The area above the dividing line contains a row of four buttons which control the display style of the various different types of objects that can be visualized in VIDA. The area beneath the dividing line is a collection of buttons whose functionality is specific to just molecules.

In the top area, the first button (with the molecule icon) contains a drop down menu of the available molecule display styles: *ball and stick*, *CPK*, *stars*, *stick*, *wireframe*, and *hidden*. Selecting one of these

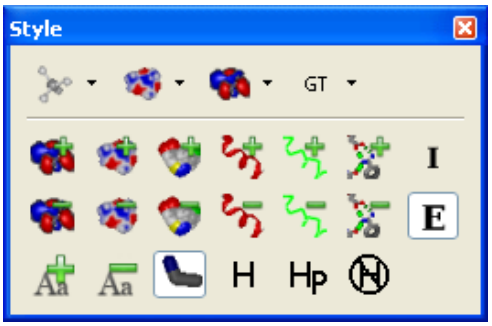


Figure 5.6: Style pane of the Style Control

options will change the display style of all the atoms and bonds in the current scope. For more details on these individual styles, see Section 5.9.1.

The second button (with the surface icon) contains a drop down menu of the available surface display styles: *solid*, *mesh*, and *points*. Selecting one of these options will change the display style of all the surfaces in the current scope. For more details on these individual styles, see section 5.11.1.

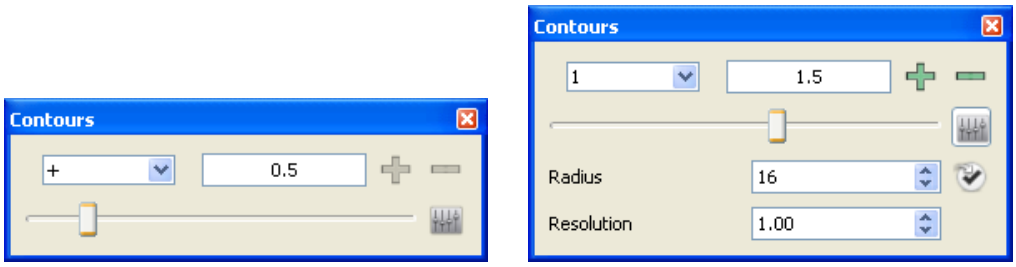
The third button (with the grid icon) contains a drop down menu of the available grid display styles: *solid*, *line*, and *cloud*. Selecting one of these options will change the display style of all the grid contours in the current scope. For more details on these individual styles, see Section 5.10.

The fourth button contains a drop down menu of the supported grid types in VIDA: *Electrostatic*, *ET*, *FRED*, *Generic*, *Difference Map*, and *Regular Map*. Selecting one of these options will change the grid type of all the grids in the current scope. A grid's type determines how it is visualized in VIDA including the default number of contours (and their levels) and the display style as well as the color of those contours. For more details on the individual grid types, see Section 5.10.1.

In the bottom area, there are three individual rows of buttons. In the first row, the first three buttons turn on the display of electrostatic grids, molecular surfaces, and accessible surfaces respectively. These specific grids and surfaces are considered display properties of their associated molecules and therefore are not displayed in the List Window (see Chapter 8). Independent (non-property) versions of these can be created from the relevant options in a right-click menu (see Section 5.1.1). The next two buttons turn on the display of protein ribbons and c-alpha traces respectively (for more information about these protein specific displays, see Section 5.9.3). The next button turns on the display of hydrogen bonds by making all of the molecules in the current scope to be hydrogen bond targets. Being a hydrogen bond target means that when it is visible, it displays any hydrogen bonds made between it and any of the other molecules that are also visible. The last button in the row toggles whether or not internal (intramolecular) hydrogen bonds are shown.

In the second row, the first three buttons turn off the display of electrostatic grids, molecular surfaces, and accessible surfaces respectively. The next two buttons turn off the display of protein ribbons and c-alpha traces respectively. The next button makes all of the molecules in the current scope no longer hydrogen bond targets. The last button in the row toggles whether or not external (intermolecular) hydrogen bonds are shown.

In the last row, the first button allows the user to specify and turn on atom and bond labels. The second button turns off all of the atom and bond labels in the current scope. The third button toggles the dis-



(a) Normal View (b) Advanced View

Figure 5.7: Contours pane of the Style Control

play of non-bonded atoms in the scene. The remaining three buttons control the hydrogen style for the molecules in the current scope. The first of the three turns all hydrogens on, the second buttons shows only polar hydrogens, and the last button hides all hydrogens.

Contours

The contour pane of the Style Control is used to control the number and level of the individual grid contours as can be seen in Figure 5.7(a). When the current application scope is set to *Focused* there will be a pulldown menu in the top left of the window containing a list of all the current grid contours. If the scope is not set to *Focused*, the scope will be displayed in place of the pulldown menu to remind users that all operations in this window apply to more than just the *Focused* grid. Next to this area is a numeric display which displays the current contour level. Next to this display are two buttons, one with a "+" icon and one with a "-" icon. These two buttons allow for the creation and deletion of individual grid contours. Please note that these options are disabled for both *Electrostatic* and *ET* grids as they can have only two contours: positive and negative. Furthermore, the contour levels of the positive and negative contours are coupled together and so changing one will change the other (but with the opposite sign).

At the bottom of this window is a slider which allows for direct control over the contour level. To the right of the slider is an *Advanced* button (with the equalizer icon) which will expand the pane to show a number of advanced options (see Figure 5.7(b)).

In the advanced section, there are two adjustable parameters (*Radius* and *Resolution*) and one button. The parameters control how contours of *Reentrant* grids are displayed according to their symmetry. The *Radius* parameter specifies how far out from the center of the scene (in Angstroms) VIDA should look in the grid in order to generate of the isocontour. The *Resolution* parameter specifies the resolution (in Angstroms) of the sub-grid used to generate the isocontours.

The single button (with the check icon) to the right of the *Radius* parameter converts the current grid contour into a fixed surface independent of the grid. This can be particularly useful when generating state files as a single surface can take up considerably less disk space than an entire grid.

Graphics

The graphics control pane in the Style Control provides an additional level of control over and interaction with the the 3D display (see Figure 5.8). It is particularly useful for users without mouse wheels.

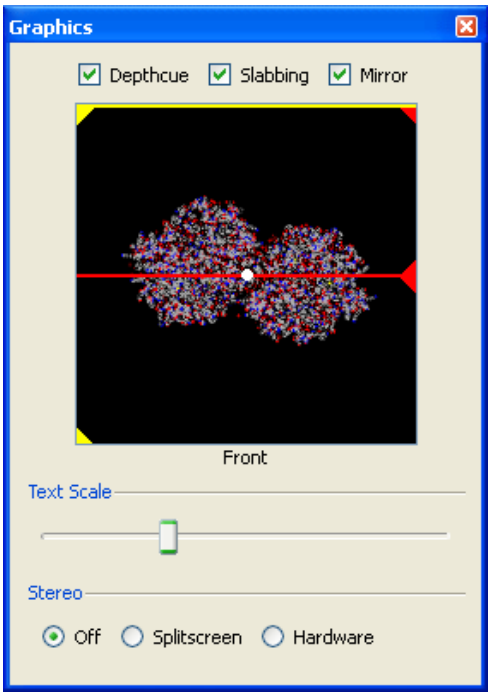


Figure 5.8: Graphics pane of the Style Control

At the top of the graphics pane are three check boxes: “Depthcue”, “Slabbing”, and “Mirror”. The first two check boxes toggle the use of their associated properties respectively. The “Mirror” check box is only enabled when slabbing is on and controls whether the front and back slabbing planes move together (mirrored around the center of the scene) or independently.

Beneath these check boxes is an interactive control which allows the user easy control over both depthcueing and the clipping planes. Displayed in the control is a top-down view of the current scene in the 3D display. The front of the scene is at the bottom as indicated by the label.

When depthcueing is enabled, two red lines will appear in the control which indicate the start and the end points in the depthcueing calculation. These lines can be moved by clicking on the triangular tab at the right of the lines and dragging them to the desired location. The 3D display will update dynamically as these lines are moved.

When clipping (or slabbing) is enabled, two yellow lines will appear in the control which indicate the position of the near and far clipping planes. These can be moved by simply clicking on the triangular tab at the left of the lines and dragging them to the desired location. The 3D display will update dynamically as these lines are moved.

In the center of the control is a small white dot that corresponds to the center of scene that the camera is pointing at. The position of the center can be changed in the Z plane by dragging the dot in the vertical plane. The position of the center can be changed in the X plane by dragging the dot in the horizontal plane.

Beneath this control is a slider which controls the scale of the text drawn in the 3D display. The 3D display will update dynamically as the slider is adjusted.

Beneath the slider are three radio buttons which control the mode of stereoscopic visualization. The default behavior is "Off". The "Hardware" option is only enabled on machines which are capable of performing 3D hardware stereo-in-a-window. This capability is determined by the computer's graphics card. For more details on stereo display, please see Section 5.3.

Custom Views

Each of the individual custom views available in VIDA can also provide their own individual panes which can be inserted into the Style Control to further enable task specific operations. These panes appear when a specific custom view is selected and disappear when the "X" icon is clicked or when another custom view is loaded. For more details on the individual custom views available, see Section 5.12.

5.2 Rendering

The images shown in the 3D display are rendered using OpenGL, therefore the quality and speed of the rendered scenes in the 3D display is dependent on the graphics card in the computer running the application. Computers with graphics cards that do not support OpenGL acceleration will still be able to display images, but the quality and speed with which the images are displayed may be markedly reduced. To help alleviate situations like this, multiple "levels of detail" are supported ranging from "Fastest" to "Best" which adjust the quality of the rendered scenes accordingly. For most computers purchased within the past year, "Best" level of detail should be appropriate.

In addition to the "level of detail", the scene can be further adjusted by modifying a few select OpenGL parameters such as the background color as well as the lighting and material models. All of these parameters can be adjusted in the application preferences (see Chapter 11).

5.3 Stereo

Both hardware and split-screen stereo displays are supported in the 3D display. Hardware stereo requires a graphics card that supports "stereo in a window" display as well as the appropriate stereo glasses. The current version of hardware stereo has been successfully tested on Microsoft Windows, Mac OS X, and SGI Irix using CrystalEyes glasses (from StereoGraphics).

In addition to the stereo-capable glasses, a CRT monitor with a high enough refresh rate is required. We recommend a refresh rate of at least 100 Hz (with 120 Hz being preferred) as the effective refresh rate of the monitor is halved due to the fact that the monitor has to swap two different scenes back and forth to create the stereo effect. Unfortunately, CRT monitors (in particular those with high refresh rates) are becoming increasingly difficult to obtain.

For those machines that do not support hardware stereo, split-sreen stereo is still a viable option. Both cross-eyed and wall-eyed views are available. Furthermore, the stereo angle and eye offset parameters

can be adjusted in the preferences (see Chapter 11).

5.3.1 Microsoft Windows

We have every expectation that most high-end workstation-class graphics cards (e.g. 3DLabs Oxygen and Wildcat, NVIDIA Quadro, ATI FireGL) will support hardware stereo but we have not been able to test all of them. Furthermore, no testing has been done on Vista and as such, no guarantees can be made about stereo visualization on Vista.

One card that we did test which caused problems was the ATI FireGL V3200. There were no problems observed with the ATI FireGL V5000.

5.3.2 Mac OS X

Mac OS X has supported "stereo in a window" visualization since late 2005 and as such hardware stereo is possible on a Mac, provided that the computer has an appropriate graphics card.

We have successfully test hardward stereo on a Mac Pro configured with the NVIDIA Quadro FX 4500 graphics card (running both Mac OS X and Microsoft Windows via Boot Camp).

5.3.3 UNIX/Linux

Currently, the only UNIX platform (that we know of) that supports hardware stereo is SGI Irix. We have successfully tested hardware stereo on a number of different SGI machines. It is important to note that in order to properly view stereo on an SGI, the screen resolution must be set to a value that is appended with the letter 's'. For instance 1200x1024 would likely not behave correctly, whereas 1200x1024s would.

5.4 Viewpoint

When viewing objects in the 3D display, there is always a center to the scene where the virtual "camera" is pointing. When the first molecule, grid, or surface is loaded into the application, the center of that object becomes the center of the scene where the camera is pointing. By default, the center does not automatically change with changes to the scene. The center can be changed explicitly from the main application toolbar, the right-click popup menu, or using the following scripting commands: `ViewerCenterSet` or `ViewerCenterAndRadiusSet`. Furthermore, the center can also be translated using the mouse interactions described in Section 5.1.1.

The center can be made to automatically change when browsing if the "Auto-Center" or "Auto-Fit" preferences are set to true (the default is false). The "Auto-Center" preference directs the window to

always recenter the scene after any changes are made to that scene. The "Auto-Fit" preference directs the window to not only recenter the scene after any changes are made but to also ensure that everything in the scene is comletely visible in the window. These preferences can be edited in the applications preferences dialog (see Chapter 11).

5.5 Bookmarks

The 3D display is capable of capturing and storing named bookmarks of the current display scene to enable viewing again at another time without having to set the scene up again by hand. Bookmarks are saved in state files and are particularly useful when trying to communicate multiple pieces of information to other users.

All bookmarks have names and are stored in order. Bookmarks can be created from the top-level *Bookmark* menu by selecting the "Add" option. The order of the bookmarks can be organized by selecting the "Organize" option in the same menu. The *Bookmark* menu also contains an "Animated" option which controls whether VIDA animates the transition between the current scene and the newly selected bookmark. A list of all the currently available bookmarks populate the rest of the menu. Bookmarks can also be created by clicking on the "Add Bookmark" button in the 3D display *Style* toolbar (see Section 5.7.1).

The 3D display also provides a bookmark display widget (see Section 5.8.2) which displays a clickable list of all the available bookmarks directly in the 3D display.

5.6 Tiled Display

The default behavior of the 3D display is to display everything that is *Visible* (see Section 3.1.2) in the same coordinate space; however, there are often occasions when it is desirable to view everything in their own space. This behavior can be obtained in "Tiled" mode, also called "Matrix" mode or "Multi-Pane" mode. "Tiled" mode can be toggled on or off from the main application toolbar.

When the 3D display is in "Tiled" mode every *Visible* and *Focused* object is displayed centered in its own individual pane. However, if desired each object can be shown relative to the same center if the "Auto-Center Panes" option in the *3D Display* section of the application preferences is unchecked. It is important to note that when "Auto-Center Panes" is enabled it is not possible to recenter or translate using mouse interactions in any of the panes.

To indicate which object is *Focused* (see Section 3.1.1), the corresponding pane will contain a blue border just inside the regular pane border. If there are any *Locked* objects (see Section 3.1.3), each of them will be displayed in every pane in addition to the individual *Visible* and *Focused* objects.

5.7 Toolbars

The 3D display provides additional toolbar areas (aside from the main application toolbar) inside the 3D window. There are four possible toolbar locations: at the top of the window, at the bottom of the window, at the left edge of the window and at the right edge of the window. Toolbars in these locations are hidden by default, but popup when the mouse is moved over the toolbar area and disappear when the mouse leaves. These toolbars can be made sticky by clicking on the button with "+" icon in it.

By default, the 3D display provides a "Style" toolbar at the top of the display and a "Mouse" toolbar at the left hand edge. Other toolbars exist in the bottom and right areas ("3D Bottom" and "3D Right" respectively) but are empty and hidden by default. They can be shown and populated via the use of scripting commands (see Chapter 10).

5.7.1 Style Toolbar

The *Style* toolbar contains a single row of buttons which are primarily used to toggle the visibility of the 3D display widgets (see Section 5.8) as well as the torn-off Style Control panes (see section 5.1.2). However, the second button (with the flagged folder icon) creates a new named display bookmark (see Section 5.5) based on the current scene and the last button toggles the use of stereoscopic visualization (see Section 5.3).

The buttons in the middle are organized into two groups. The first group contains four buttons which toggle the display of the annotation widget, the 2D depiction widget, the data display widget, and the bookmark widget respectively. The second group contains five buttons which toggle the display of the torn-off Style Control panes: color, selection, style, contours, and graphics respectively.

This toolbar can be referenced in scripting commands using the name "Style".

5.7.2 Mouse Toolbar

The *Mouse* toolbar contains a single row of buttons which control the behavior of the mouse in the 3D display. Ignoring the "sticky" button, the first button puts the mouse into the default mode which obeys all of the expected interaction mechanisms as described in Section 5.1.1. The second button (with the info icon) changes the behavior of the mouse such that an informative label is displayed on the screen regarding whatever is currently underneath the mouse cursor. The remaining three buttons put the mouse into one of three measurement modes: *Distance*, *Angle*, and *Torsion* respectively. More details about measurement and the monitors generated by the measurements can be found in Section 5.9.4.

This toolbar can be reference in scripting commands using the name "Mouse".

5.7.3 Application Toolbar

In addition to providing its own toolbars, the 3D display is affected by a few of the buttons in the main application toolbar. The application toolbar contains the "Tiled" display toggle button (matrix icon), the main window screenshot button (camera icon), the centering button (four arrows pointing in), and the fitting button (four arrows pointing out).

The "Tiled" display button toggles the display between single pane overlay mode and multiple pane tiled mode. For more details see Section 5.6. The screenshot button captures an image of the current main window at a user specified resolution. The centering button centers the display on the currently selected set or whatever molecule(s) are in the current scope. The fit button centers the scene and adjusts the scale factor such that all *Visible* objects can be seen on the screen at the same time.

5.8 Display Widgets

The 3D display provides a number of additional widgets which are drawn directly in 3D display as opposed to on the side or floating above it. These widgets include an annotation widget, a bookmark widget, a data display widget, and a depiction widget.

5.8.1 Annotation

It is often desirable when viewing molecules to make notes or simple annotations about specific molecules. This can be done directly within the 3D display, where the annotation widget appears like a transparent Post-It™ note (see Figure 5.9). The color of the widget as well as the font used can be specified in the application preferences.

The widget can be moved by clicking on the title bar and dragging it to the desired location. The widget can be resized by grabbing the small triangular tab in the lower right-hand corner of the widget and dragging until the desired size is achieved.

Clicking in the annotation widget will activate it for editing. Editing can be accomplished by directly typing once the widget has been clicked or by clicking on the "Edit" button in the title bar. Clicking on the "Edit" button will launch a more fully featured text editor which supports copy and paste operations (see Figure 5.10).

Annotations are also stored in the spreadsheet (see Chapter 7) and can be edited there as well as in the 3D display. Annotations are saved as a SD tag when a molecule is written out (in either SD or OEB formats).

5.8.2 Bookmark

The bookmark widget is a simple widget drawn at the bottom of the 3D display containing a clickable list of the currently available bookmarks. Clicking on any individual bookmark will load that bookmark and update the display accordingly. For more information on bookmarks, see Section 5.5.

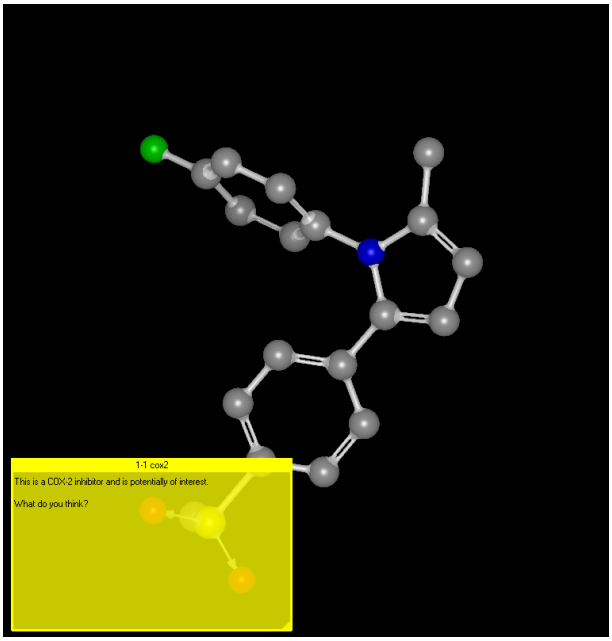


Figure 5.9: Molecule with annotation

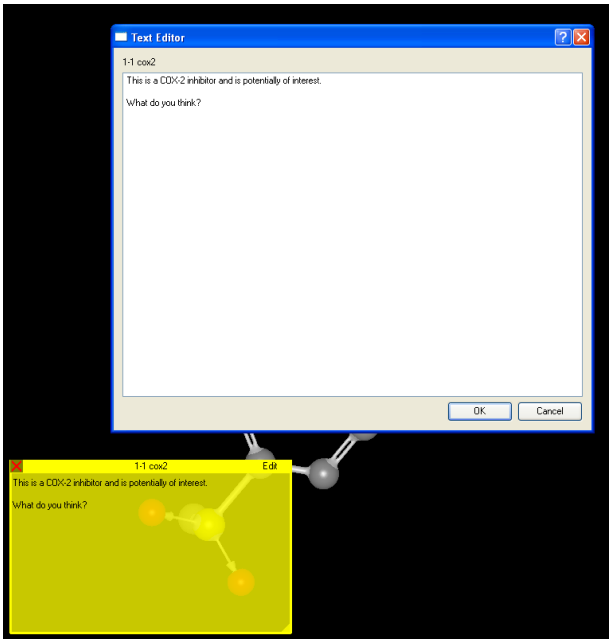


Figure 5.10: Editing the annotation

5.8.3 Data Display

This widget provides a simple table which contains the spreadsheet data of the *Focused* object. The widget can be moved around the 3D display by clicking and dragging it to the desired location. It can also be resized by clicking in the tab in the upper left hand corner. Lastly, if the amount of data associated with the object is too large to be shown all at once, the widget does provide scrolling functionality.

5.8.4 Depiction Widget

The depiction widget draws a 2D depiction of the focused molecule directly into the 3D display. The depiction is drawn in the lower left hand corner of the display (or individual pane if in tiled mode). The depiction style is determined based on the application 2D depiction preferences. The size and other drawing parameters (e.g. anti-aliasing) of the depiction are also specified in the applicaton preferences.

5.9 Molecular Visualization

This section describes the very powerful, versatile, and interactive molecular visualization capabilities of the 3D display. The interactive control of molecular visualization is typically performed using the *Style Control* (see Section 5.1.2) or through scripting commands (for more details on scripting see Chapter 10).

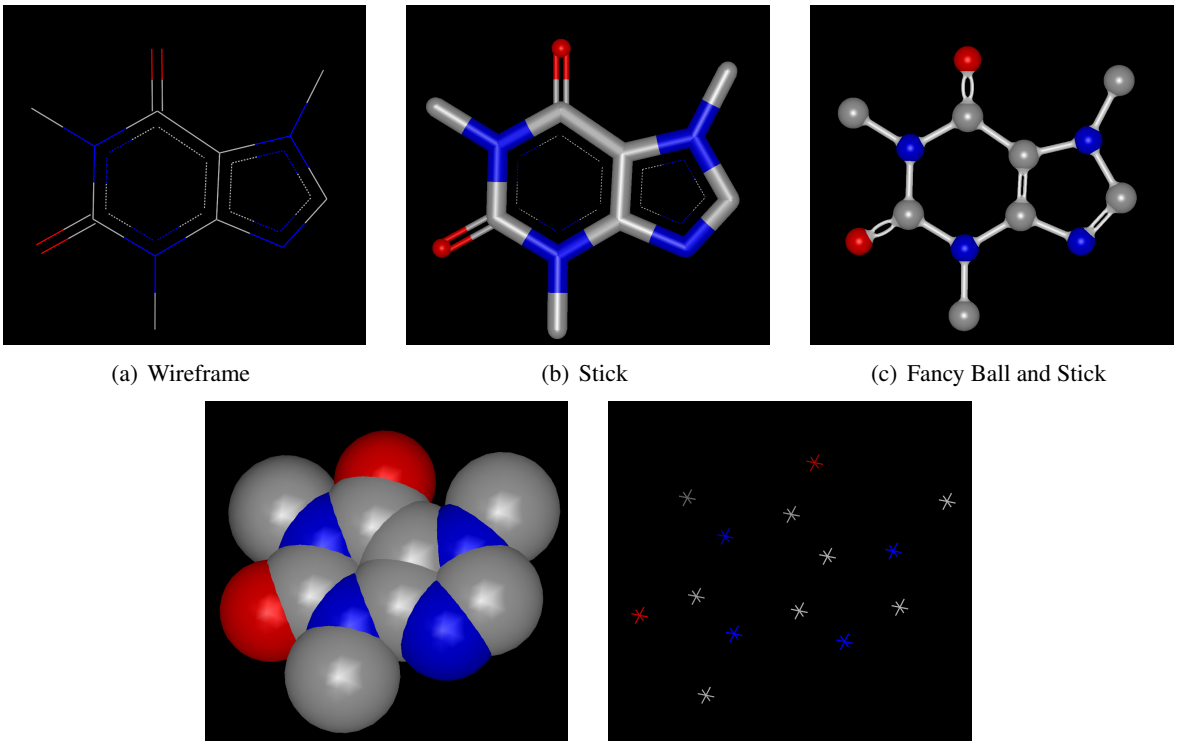
5.9.1 Display Styles

All of the standard display styles are available (*Wireframe*, *Stick*, *Ball and Stick*, *CPK*, and *Stars*) and can be seen in Figure 5.11.

By default, small molecules are drawn in *Stick* mode while large molecules are drawn in *Wireframe* mode. These defaults can be changed in the application preferences (see Chapter 11).

5.9.2 Color

By default, the atoms in a molecule are colored according to their element types and the bonds in a molecule are colored according to the element types of the two atoms defining that bond. There are two color palettes that specify the actual colors used for a given element, one for use with dark-colored backgrounds and the other for use with light-colored backgrounds. Both of these palettes can be viewed and modified in the application preferences (see Chapter 11). The dark-background atom color palette can be seen in Figure 5.12(a).



(a) Wireframe (b) Stick (c) Fancy Ball and Stick

(d) CPK (e) Stars

Figure 5.11: Molecule rendering styles

Atom Colors
Default
Restore
Ok
Cancel
Residue Colors
ALA ARG ASN ASP CYS GLN GLU GLY HIS ILE
LEU LYS MET PHE PRO SER THR TRP TYR VAL
UNK ASX GLX CYX CYH HID HIE HIP
A C G T U
ABA ACE CGU CME CSD HSE HYP HYL MLE MLY
MSE ORN PCA PTR SAR SEP TAU TPO
CL BR EOH GOL HOH IOD MOH PER PO4 SO4
Default
Restore
Ok
Cancel

(a) Atom Color Palette (b) Residue Color Palette

Figure 5.12: Dark background atom and residue color palettes

Coloring Schemes

There are a number of molecule specific coloring schemes in addition to the standard element based coloring. The following schemes are discussed below: amino, bfactor, carbon, chain, cpk, cpknew, element, formal charge, group, partial charge, reference, residue, and shapely.

Amino

The *amino* color scheme is a protein specific scheme which colors atoms according to their individual residues. The residue colors are listed below.

ASP, GLU	Bright Red	[230, 10, 10]
CYS, MET	Yellow	[230, 230, 0]
LYS, ARG	Blue	[20, 90, 255]
SER, THR	Orange	[250, 150, 0]
PHE, TYR	Mid Blue	[50, 50, 170]
ASN, GLN	Cyan	[0, 220, 200]
GLY	Light Grey	[235, 235, 235]
LEU, VAL, ILE	Green	[15, 130, 15]
ALA	Dark Grey	[200, 200, 200]
TRP	Purple	[180, 90, 180]
HIS	Pale Blue	[130, 130, 210]
PRO	Flesh	[220, 150, 130]
Others	Tan	[190, 160, 110]

BFactor

The *bfactor* color scheme is a protein specific scheme which colors atoms according to the bfactor value using a fixed blue-to-red gradient between 0 and 100.

Carbon

The *carbon* color scheme colors all of the carbon atoms in the current scope the same color which is specified by the user.

Chain

The *chain* color scheme is a macromolecule specific scheme which assigns a unique color to each of the macromolecular chains.

CPK, CPKNew

The *CPK* color scheme is an element specific scheme based on the colors used in the popular CPK plastic space filling models. The *CPKnew* color scheme is a variant on the *CPK* color in that it uses a slightly brighter version of some of the colors. The atom colors are listed below with the RGB values for *CPK* in the third column and the RGB values for *CPKNew* in the fourth column (if different).

H	White	[255, 255, 255]	
He	Pink	[255, 192, 203]	
Li	Fire Brick	[178, 34, 34]	[178, 33, 33]
B,Cl	Green	[0, 255, 0]	
C	Light Grey	[200, 200, 200]	[211, 211, 211]
N	Sky Blue	[143, 143, 255]	[135, 206, 235]
O	Red	[255, 0, 0]	
F,Si,Au	Golden Rod	[218, 165, 32]	
Na	Blue	[0, 0, 255]	
Mg	Forest Green	[34, 139, 34]	
Al,Ca,Ti,Cr,Mn,Ag	Dark Grey	[128, 128, 144]	[105, 105, 105]
P,Fe,Ba	Orange	[255, 165, 0]	[255, 170, 0]
S	Yellow	[255, 200, 50]	[255, 255, 0]
Ni,Cu,Zn,Br	Brown	[165, 42, 42]	[128, 40, 40]
I	Purple	[160, 32, 240]	
Unknown	Deep Pink	[255, 20, 147]	[255, 22, 145]

Element

The *element* color scheme is the default color scheme used when coloring molecules. Colors are assigned according to atomic number. The actual colors used for a given element can be edited in the application preferences as seen in Figure 5.12(a).

H	White	[255, 255, 255]
C	Grey	[180, 180, 180]
N	Blue	[0, 0, 255]
O	Red	[255, 0, 255]
F	Green	[0, 255, 0]
P	Magenta	[192, 0, 192]
S	Yellow	[255, 255, 0]
Cl	Lime Green	[170, 255, 0]
Br	Dark Red	[170, 0, 0]
I	Dark Orange	[170, 85, 0]
Group I	Cyan	[0, 255, 255]
Group II	Light Blue	[160, 160, 255]
Transition Metals	Purple	[170, 0, 255]
Others	Pink	[255, 0, 128]

Formal Charge

The *formal charge* color scheme colors atoms according to their formal charge using a fixed red-to-blue gradient between -4 and +4.

Group

The *group* color scheme is a macromolecule specific scheme which colors each atom according to its position in a macromolecular chain. The colors are assigned along a smooth rainbow spectrum from blue to green to yellow to orange to red.

Partial Charge

The *partial charge* color scheme colors atoms according to their partial charge using a red-to-blue gradient between -1.0 and +1.0.

Reference

The *reference* color scheme colors all of the carbon atoms in the current scope using the current *Reference* color which can be set in the application preferences. The default is green.

Residue

The *residue* color scheme is a protein specific scheme which colors atoms according to their individual residues. The actual colors used can be edited in the application preferences as seen in Figure 5.12(b). The default colors correspond to those used in the *shapely* color scheme.

Shapely

The *shapely* color scheme is a protein specific scheme which colors atoms according to their individual residues. This scheme is based upon Bob Fletterick's "Shapely Models" [1]. The residue colors are listed below.

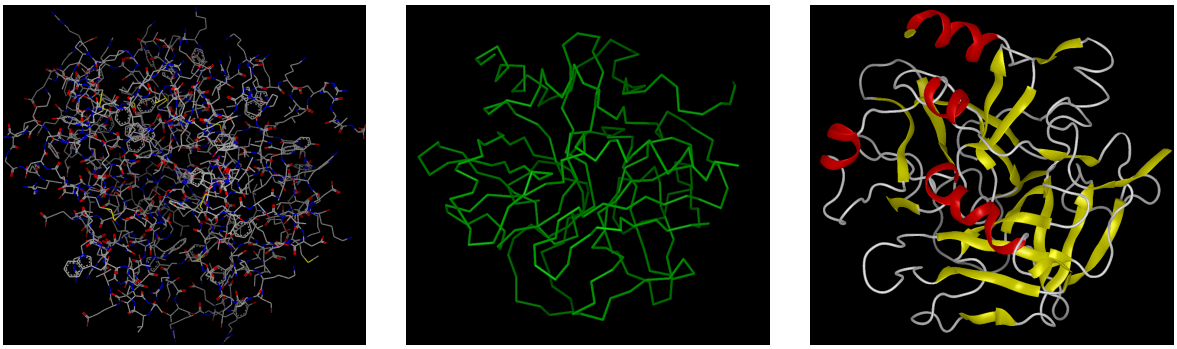
ALA	Medium Green	[140, 255, 140]
GLY	White	[255, 255, 255]
LEU	Olive Green	[69, 94, 69]
SER	Medium Orange	[255, 112, 66]
VAL	Light Purple	[255, 140, 255]
THR	Dark Orange	[184, 76, 0]
LYS	Royal Blue	[71, 71, 184]
ASP	Dark Rose	[160, 0, 66]
ILE	Dark Green	[0, 76, 0]
ASN	Light Salmon	[255, 124, 112]
GLU	Dark Brown	[102, 0, 0]
PRO	Dark Grey	[82, 82, 82]
ARG	Dark Blue	[0, 0, 124]
PHE	Olive Grey	[83, 76, 66]
GLN	Dark Salmon	[255, 76, 76]
TYR	Medium Brown	[140, 112, 76]
HIS	Medium Blue	[112, 112, 255]
CYS	Medium Yellow	[255, 255, 112]
MET	Light Brown	[184, 160, 66]
TRP	Olive Brown	[79, 70, 0]
ASX,GLX,PCA,HYP	Medium Purple	[255, 0, 255]
A	Light Blue	[160, 160, 255]
C	Light Orange	[255, 140, 75]
G	Medium Salmon	[255, 112, 112]
T	Light Green	[160, 255, 160]
Backbone	Light Grey	[184, 184, 184]
Special	Dark Purple	[94, 0, 94]
Default	Medium Purple	[255, 0, 255]

5.9.3 Proteins

In addition to the standard display styles, the two protein specific displays of *C-Alpha Traces* and *Ribbons* are supported. Examples of these display styles can be seen in Figure 5.13.

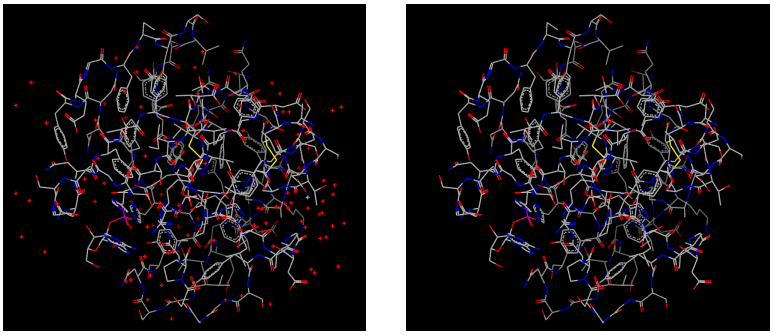
Given that protein files often inevitably contain many extra components that are not necessarily of interest to the user (waters for example), the ability to hide the display of non-bonded atoms is provided to ease the viewing of proteins without having to manually edit the input file. An example of a protein displayed with and without waters can be seen in Figure 5.14.

Another common scenario when working with proteins is that the input file contains a large multimer which as expected is interpreted as a single molecule. However, sometimes the ability to interact with the individual components is of particular value. The ability to split a molecule into separate components can be performed by selecting the "Components" option in the *From Split* submenu in the *New Molecule* submenu in the top-level *File* menu. This function will then determine what the individual components of the *Focused* molecule are and create a new list containing a new molecule for each component. The



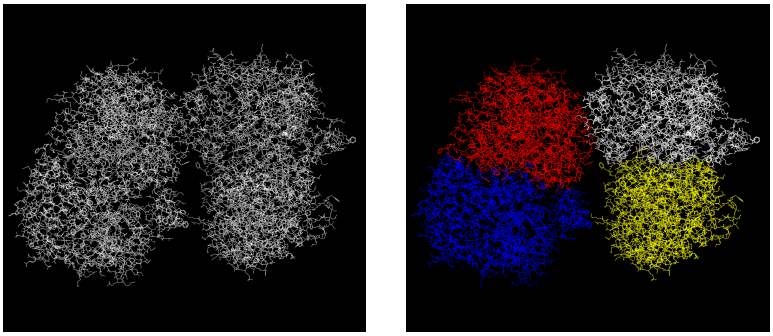
(a) Wireframe (b) C-Alpha Trace (c) Ribbon

Figure 5.13: Different views of the same protein



(a) Protein with Waters (b) Protein without Waters

Figure 5.14: Same protein with and without waters



(a) Multimer displayed as a single molecule (b) Multimer displayed as multiple molecules

Figure 5.15: Same protein as a single molecule vs as multiple molecules

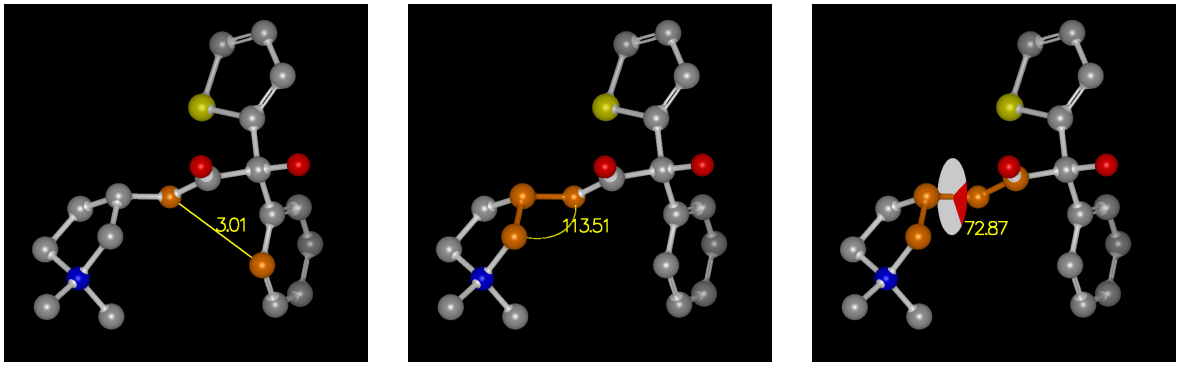
original molecule is not changed or deleted. An example of this process can be seen in Figure 5.15. The *From Split* submenu also contains "Selected" and "Marked" options which will create new molecules from the original based on the selected or marked set of atoms.

It may also be of interest to put some of these components back together into a single molecule. This can be done by choosing one of the following options from the *From Merge* submenu in the *New Molecule*

in the top-level *File* menu: “Selected”, “Marked”, “Visible”. Choosing any one of these options will create a new molecule which is composed of all the molecules that were either “Selected”, “Marked”, or “Visible” at the time of the operation.

5.9.4 Monitors

The ability to display measurement monitors is a useful feature and is well supported. Access to these measurement facilities is available from the *Mouse* popup toolbar on the left hand side of the 3D display. There are three buttons which turn on *Distance*, *Angle*, and *Torsion* measurement respectively.



(a) Distance Monitor (b) Angle Monitor (c) Torsion Monitor

Figure 5.16: Different types of monitors

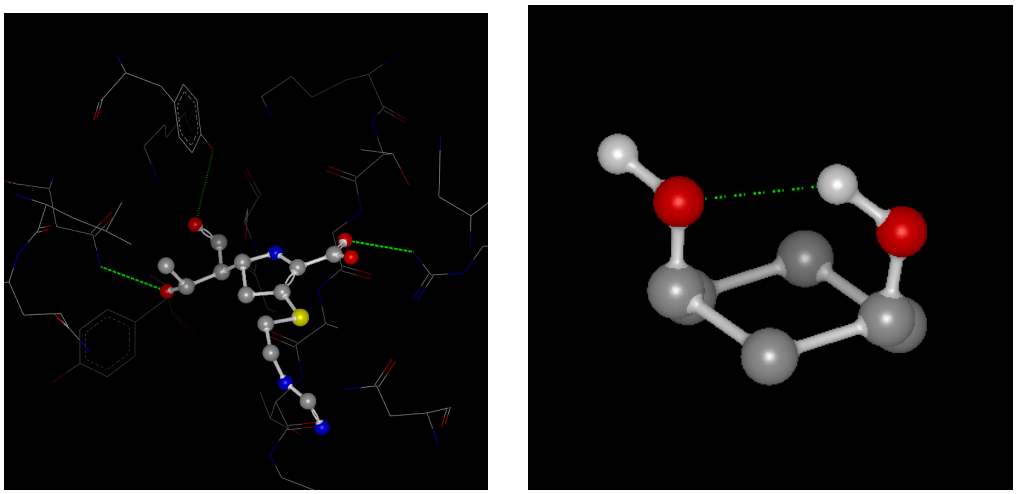
When the mouse is put into one of these modes, the measurements are made based on the selected set of atoms. In *Distance* mode, the first atom selected is consider the “anchor” atom. Once the anchor atom has been selected, a temporary distance monitor will be displayed to any other atom that the mouse passes over. Selecting another atom will create a permanent monitor and clear the anchor. In *Angle* mode, the first two atoms selected are considered the anchor atoms and a temporary angle monitor will be displayed to any other atom that the mouse passes over. Selecting a third atom will create a permanent monitor and clear the anchors. In *Torsion* mode, the first three atoms selected are considered the anchor atoms and a temporary torsion monitor will be displayed to any other atom that the mouse passes over. Selecting a fourth atom will create a permanent monitor and clear the anchor.

Monitors can be removed by performing a right-click operation in the 3D display and selecting the “Delete visible monitors” option. Monitors are also displayed in the “List Window” and can be deleted individually there (see Chapter 8). Examples of the three types of monitors can be seen in Figure 5.9.4.

5.9.5 Hydrogen Bonds

Both internal (intramolecular) and external (intermolecular) hydrogen bonds can be displayed. Hydrogen bonds are determined by typing all of the atoms as either acceptors, donors, both, or neither. Then the hydrogen bond energy is calculated between all acceptors and donors based on the ChemScore [2]

scoring function which takes into account both interatomic distance as well as the coordination geometry. This function provides a relative score between 0 and 1.0 for the energy of the hydrogen bond. For all hydrogen bonds with an energy greater than 0, the bond is drawn as a dotted green line. The thickness of the line as well as the spacing between the dots correlates to the bond energy. Therefore, thicker more solid lines have higher energy than thinner more dotted lines. An example of both internal and external hydrogen bonds can be seen in Figure 5.17.



(a) External Hydrogen Bonds (b) Internal Hydrogen Bonds

Figure 5.17: Hydrogen bonds in a protein active site

5.10 Grid Visualization

Grids are spatial arrays of data that are sampled at regular intervals. Grids may contain electrostatic samples of spatial regions, density samples, and other potential fields.

A grid can have a variable number of contours. Each contour has an index starting at 1 for the first contour. Additionally, every contour has its own individual color and a threshold value which can be positive or negative (also known as the "contour level"). There are three different display styles for grids contours: *Solid*, *Mesh* and *Cloud*.

5.10.1 Grid Types

As mentioned in the previous section, there are multiple different predefined grid types that are supported including: *Electrostatic*, *ET*, *FRED*, *Generic*, *Difference Map*, and *Regular Map*.

Electrostatic

Electrostatic grids are the only type of grid that can be actually be generated within VIDA as opposed to simply being visualized. Electrostatic grids are created for individual molecules using OpenEye's Zap toolkit. The creation of the grid requires the presence of partial charges on the input molecule. If partial charges were already present on the molecule, those will be used in the calculation. However, if no partial charges were present (or the user decides to ignore them, which can be done in the application preferences), temporary partial charges will be calculated using MMFF94 or AM1-BCC [3, 4]. If a molecule has greater than 50 atoms, MMFF94 charging will be done even if AM1-BCC was selected. However, for proteins an alternate residue based charging model is available and can be selected in the preferences. This model assigns partial charges based on residue information as is shown below:

ASP	OD1, OD2	-0.5
GLU	OE1, OE2	-0.5
LYS	NZ	+1.0
ARG	NE	+1.0
Other		0.0

The grid resolution is dynamically determined based on the number of atoms in the input molecule. However, it is possible to specify a fixed resolution in the grid preferences. Be warned, however, that specifying very fine resolutions (<2.0 Angstroms) may consume a surprisingly large amount of memory, and in some cases more than may be available on the computer, which can lead to program failure. This should be consider a very advanced feature to be used only when necessary and when all other data has been saved. Finally, it has been observed in research at OpenEye that adding salt to the electrostatic calculation has a good effect on the grids generated. The salt concentration can be specified in the preferences. The default concentration is 0.04 M. The maximum concentration that can be specified is 0.1 M.

Electrostatic grids (as well as ET grids) are special in that they are always generated with two default contours: a positive one and a negative one. These contours cannot be deleted and new contours cannot be added. The positive contour is colored blue by default and the negative contour is colored red. The contour levels of the two contours are coupled together and so changing one will change the other.

ET

ET (or Electrostatic Tanimoto) grids are generated by OpenEye's Brood and EON applications. When viewed in VIDA, they behave and are treated essentially the same as Electrostatic grids (see Section 5.10.1).

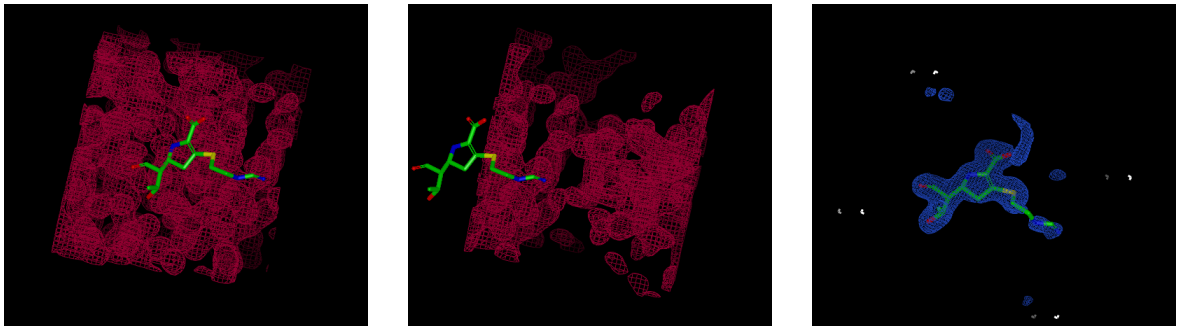
FRED

FRED grids are shape potential grids that are created by OpenEye's FRED application. In VIDA, the behave just like Generic grids (see Section 5.10.1) except that their default contour color is purple instead of blue.

Generic

Generic grids are effectively the most basic grids and act as the primary grid workhorses. They are completely customizable and have bare-bones settings. They default to having just one contour with a default color of blue. Additional contours can be added as well as removed. These default values can be changed in the *Grids* section of the application preferences.

5.10.2 Reentrant Grids



(a) Reentrant (b) Reentrant (c) Non-reentrant

Figure 5.18: Grid contouring examples showing (a) an infinite grid (b) the same grid rendered at a different center, notice that the grid is rendered in the center of the scene, not at the molecule and (c) a finite density grid showing the grid corners or extent of the grid.

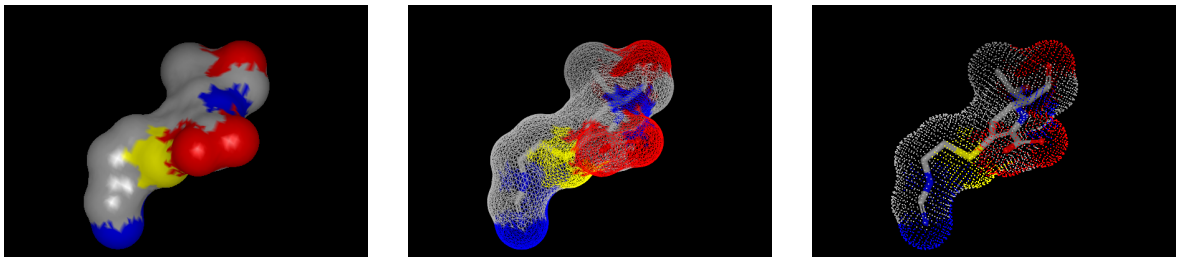
Some grids have fixed extents and some grids are infinitely reentrant in space. Reentrant grids obviously cannot be displayed in their entirety, so instead only the region around the center of the current scene is displayed. Grids that are being rendered as reentrant can be distinguished from non-reentrant grids by the absence of grid corners which define the extents of non-reentrant grids.

5.11 Surface Visualization

Surfaces are infinitely thin three dimensional connected regions that represent objects such as molecular or accessible surfaces of molecules. Surfaces can be selected (or scribed), cropped, colored, and made transparent in part or in whole.

5.11.1 Display Styles

Surfaces can be visualized in one of three display styles: Solid, Mesh, and Points. These individual style can be seen in Figure 5.19.



(a) Solid (b) Mesh (c) Points

Figure 5.19: Surface rendering examples showing (a) solid rendering, (b) mesh rendering, and (c) point rendering.

5.11.2 Color

There are a number of surface specific coloring schemes in addition to the standard single color scheme. The following schemes are discussed below: atom color, concavity, curvature, distance, electrostatics, grid, hydrogen bond potential, hydrophobicity, and surface potential.

Atom Color

The *atom color* scheme colors each vertex on the surface using the color of the nearest atom to that vertex in the molecule that the surface was created from (see Figure 5.20(b)). This scheme will not work if the surface was not created from a molecule and also if that molecule is not currently present.

Concavity

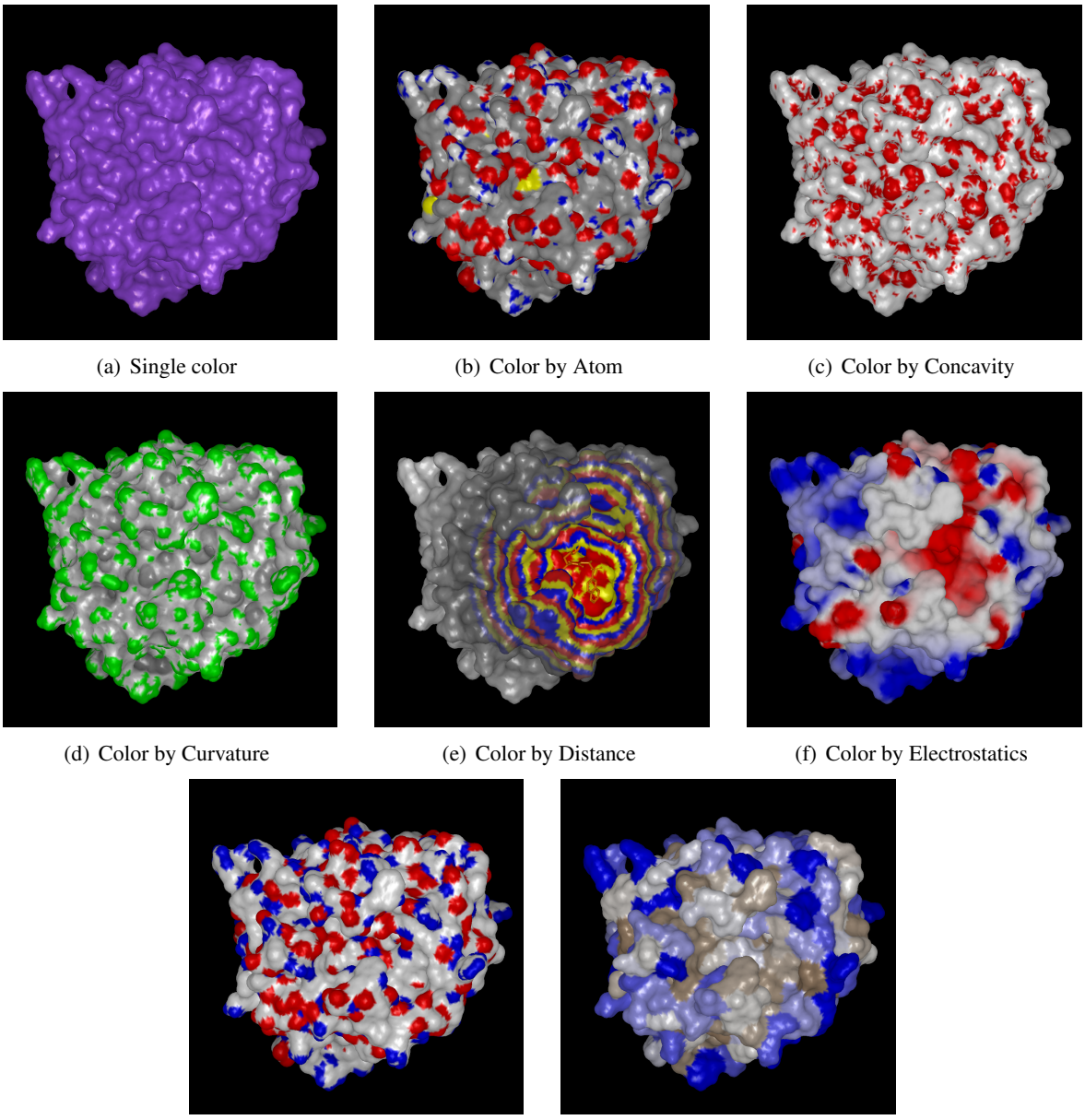
The *concavity* scheme colors each vertex on the surface according to how concave the surface is at that vertex using a red-to-white gradient where red indicates a region of high concavity (see Figure 5.20(c)).

Curvature

The *curvature* scheme colors each vertex on the surface according to the curvature of the surface at that vertex using a grey-to-green gradient where green indicates a region of high curvature (see Figure 5.20(d)).

Distance

The *distance* scheme colors each vertex by its distance from either the selected set of atoms or the visible set of atoms (if none are selected at the time of coloring). Each band of color represents a distance of one Angstrom from the selected set. The color banding is continuously repeated as the distance increases, but the colors fade to white with each repetition to show the distance effect (see Figure 5.20(e)).



(a) Single color (b) Color by Atom (c) Color by Concavity

(d) Color by Curvature (e) Color by Distance (f) Color by Electrostatics

(g) Color by Hydrogen Bond Potential (h) Color by Hydrophobicity

Figure 5.20: Various different surface coloring schemes

Electrostatics

The *electrostatics* scheme colors each vertex on the surface according to the electrostatic potential at that vertex using a red-to-blue gradient from -7.0 to +10.0 (see Figure 5.20(f)). This range can be changed if desired in the application preferences in the *Surfaces* section.

The electrostatic potential at the surface is calculated using OpenEye's Zap toolkit (for more information, please see the Zap toolkit documentation). By default, VIDA will use the molecule's input partial charges

in the calculation (if any were specified). However, if no partial charges were specified, VIDA will assign partial charges using either MMFF94 (the default) or AM1-BCC [3, 4]. The choice of charge model can be specified in the application preferences. If a molecule has greater than 50 heavy atoms, MMFF94 will always be used in place of AM1-BCC regardless of the set preferences.

Proteins have an alternative charging option based on residue information. The charge model can be seen in the table below. This model is used by default (if no partial charges were specified), but it can be disabled in the application preferences.

ASP	OD1, OD2	-0.5
GLU	OE1, OE2	-0.5
LYS	NZ	+1.0
ARG	NE	+1.0
Other		0.0

Grid

The *grid* scheme colors each vertex on the surface according to the grid potential at that vertex using a red-to-blue gradient from the minimum grid value to the maximum grid value. The grid potential is determined by mapping the specified grid onto the surface.

Hydrogen Bond Potential

The *hydrogen bond potential* scheme colors each vertex according to the hydrogen bond class of the nearest atom to that vertex in the molecule that the surface was created from (see Figure 5.20(g)). This scheme will not work if the surface was not created from a molecule and also if that molecule is not currently present.

Hydrophobicity

The *hydrophobicity* scheme colors each vertex according to a specified hydrophobic color scale based on residue information associated with a given vertex. This scheme will not work if the surface was not created from a protein and also if that protein is not currently present.

There are four different scales available: Charifson, Eisenberg, Kyte-Dolittle, and White Octanol. The Charifson scale applies defined colors based on residue information. The other three scales each return a specific hydrophobicity value which is then used to return a color based on a brown-to-blue gradient between -2.5 and +2.5 for Eisenberg, -4.5 and 4.5 for Kyte-Dolittle, and -4.0 and 4.0 for White Octanol.

Charifson

All	CA,CB,CG,CD	Yellow	[0, 255, 255]
ASP		Red	[255, 0, 0]
GLU		Red	[255, 0, 0]
ASN		Magenta	[192, 0, 192]
GLN		Magenta	[192, 0, 192]
HIS		Magenta	[192, 0, 192]
LYS		Blue	[0, 0, 255]
ARG		Blue	[0, 0, 255]
ALA		Yellow	[255, 255, 0]
PHE		Yellow	[255, 255, 0]
ILE		Yellow	[255, 255, 0]
GLY		Yellow	[255, 255, 0]
LEU		Yellow	[255, 255, 0]
PRO		Yellow	[255, 255, 0]
VAL		Yellow	[255, 255, 0]
MET		Yellow	[255, 255, 0]
CYS	S	Magenta	[192, 0, 192]
CYS	!S	Yellow	[255, 255, 0]
TRP	N	Magenta	[192, 0, 192]
TRP	!N	Yellow	[255, 255, 0]
SER	O	Magenta	[192, 0, 192]
SER	!O	Yellow	[255, 255, 0]
THR	O	Magenta	[192, 0, 192]
THR	!O	Yellow	[255, 255, 0]
TYR	O	Magenta	[192, 0, 192]
TYR	!O	Yellow	[255, 255, 0]
Other		White	[255, 255, 255]

	Eisenberg	Kyte-Dolittle	White Octanol
ALA	0.62	1.80	-0.50
ARG	-2.53	-4.50	-1.81
ASN	-0.78	-3.50	-0.85
ASP	-0.90	-3.50	-3.64
CYS	0.29	2.50	0.02
GLN	-0.85	-3.50	-0.77
GLU	-0.74	-3.50	-3.63
GLY	0.48	-0.40	-1.15
HIS	-0.40	-3.20	-2.33
ILE	1.38	4.50	1.12
LEU	1.06	3.80	1.25
LYS	-1.50	-3.90	-2.80
MET	0.64	1.90	0.67
PHE	1.19	2.80	1.71
PRO	0.12	-1.60	-0.14
SER	-0.18	-0.80	-0.46
THR	-0.05	-0.70	-0.25
TRP	0.81	-0.90	2.09
TYR	0.26	-1.30	0.71
VAL	1.08	4.20	0.46
ASX	-0.84	-3.50	-2.25
GLX	-0.80	-3.50	-2.20
Other	0.00	-0.49	-0.52

Surface Potential

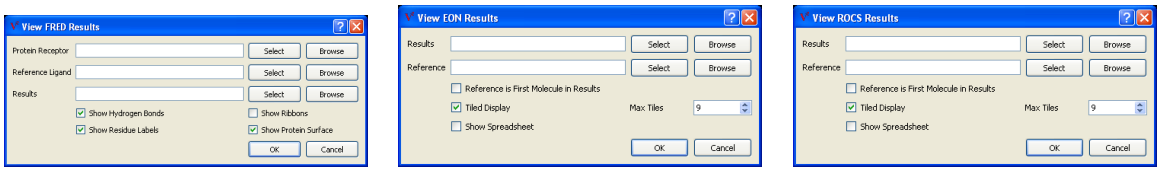
The *surface potential* scheme colors each vertex according to the potential value at that vertex using a red-to-blue gradient from the surface potential minimum to the surface potential maximum. This scheme is different than the *electrostatic* scheme in that the coloring is based on the stored potential values as opposed to calculated potential values. Typically, surfaces generated within VIDA will not have any stored potential values. However, surfaces created externally using OpenEye's Spicoli toolkit for example can store values in the potential field and as such can be colored accordingly.

5.11.3 Selection

Vertices can be selected simplying by clicking on the desired location. A line of vertices can be selected by selecting one vertex and then by holding down the *Shift* key when selecting another. This starting selection can be grown (and shrunk) using the Selection pane in the Style Control (see Section 5.1.1). Double-clicking on the surface will select the entire surface.

5.12 Custom Views

There are a number of very common uses of the 3D display which can take some effort to set up correctly. For this reason, a number of automatically setup custom views are available. These custom views are designed to quickly show in an expected and logical manner the results of various different applications. Support currently exists for viewing FRED (docking), ROCS, or EON results. These views can be accessed by choosing the desired option in the *Tools* menu. An example of the interface to each of these custom views can be seen in Figure 5.21.



(a) FRED Results View Window (b) EON Results View Window (c) ROCS Results View Window

Figure 5.21: The Custom Views Dialogs

The first thing to notice about all the views are the molecule choice buttons. The *FRED View* has three, while the *EON* and *ROCS Views* have only two. Clicking on the *Select* button next to any of these allows selection of a molecule from the currently loaded set. Clicking on the *Browse* button allows selection of a molecule from a file on disk. Following the molecule choice buttons are various check boxes for customizable display options for that view.

5.12.1 FRED View

The first view is the *FRED View*, the results of which can be seen in Figure 5.23(a). This view sets up the protein's active site (the region around the reference ligand) with a cropped electrostatic surface and the nearby molecular structure. Optionally, it will also display the protein's associated ribbon. Finally, it labels the c-alpha carbons on the protein with residue information.

The *FRED View* then places the reference ligand in the active site and colors the carbons using the "Reference Color" (the default is green). Finally, it loads the optional results and makes the first molecule in that list the *Focused* molecule.

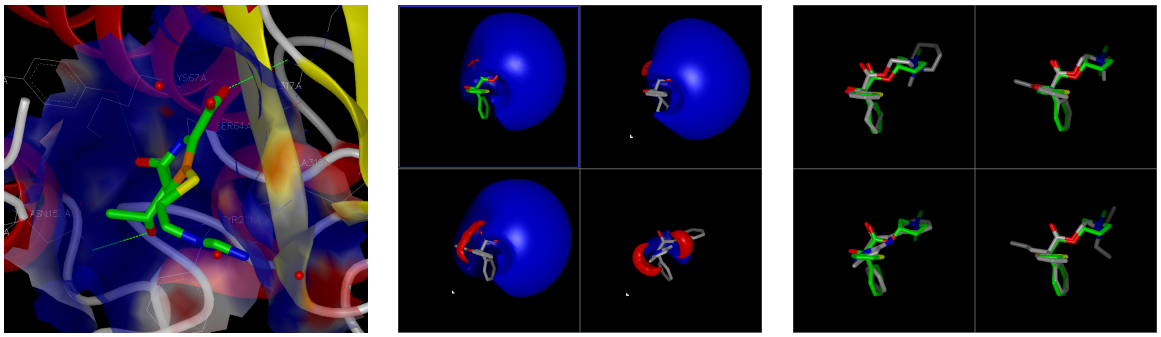
The *FRED View* has several optional parameters, as shown in Figure 5.21. Hydrogen bonds, protein ribbons, residue labels, and the protein surface can all be toggled using the checkboxes at the bottom of the dialog. Furthermore, there are additional preferences with respect to the distance cutoffs used that

Figure 5.22: Custom style pane for FRED View

can be found in the application preferences (see Chapter 11).

As mentioned in Section 5.1.2, the *FRED View* adds a custom pane to the Style Control to enable more specific interactions with the scene generated by *FRED View* (see Figure 5.22).

The custom pane contains a row of nine buttons above a "Setup" button which relaunches the original *FRED View* dialog (see Figure 5.21(a)). The first eight buttons toggle the display of hydrogen bonds to the protein, residue labels, ribbons, active site surface, the FRED inner docking contour, the FRED outer docking contour, the reference ligand, and any other extra molecules that might be associated with the protein such as waters or co-factors. The last button toggles the display of FRED docking constraints (if any) and furthermore contains a drop down menu with a list of all the individual constraints.



(a) A FRED Results View (b) An EON Results View (c) A ROCS Results View

Figure 5.23: The Custom Views

5.12.2 EON View

The *EON View* has two molecule pickers, one for the result set, and one for the optional reference molecule. Below these are the parameters to the *EON View*. The first checkbox specifies whether or not to use the first result as the reference molecule or not. The second checkbox is the Tiled display option, which sets the application into Tiled mode with the various results displayed in their own individual panes. Next to this checkbox is a numeric entry box to set the max number of tiles in the display. Finally, the last check box toggles whether or not to load the spreadsheet display as well (see Chapter 7).

The *EON View* places the reference molecule in the first pane and colors that molecule using the "Reference Color" (the default is green). The other molecules are displayed in their own panes. Each molecule has its electrostatic grid calculated and displayed for comparison with the reference. An example of this can be seen in Figure 5.23(b).

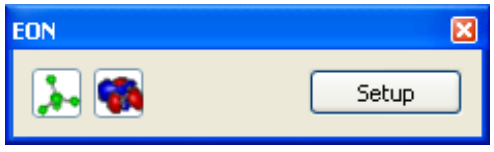


Figure 5.24: Custom style pane for EON View

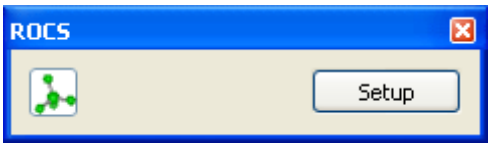


Figure 5.25: Custom style pane for ROCS View

As mentioned in Section 5.1.2, the *EON View* adds a custom pane to the Style Control to enable more specific interactions with the scene generated by *EON View* (see Figure 5.24).

The custom pane contains a row of two buttons followed by a "Setup" button which relaunches the original *EON View* dialog (see Figure 5.21(b)). The first button toggles the display of the reference molecule. The second button toggles the display of any ET grids that were attached to the molecules by EON.

5.12.3 ROCS View

The last custom view is the *ROCS View*. This view, like the *EON View* (see Section 5.12.2), has two molecule pickers, one for results and one for a reference molecule. It also has all of the same options as the EON view, plus an additional checkbox to specify whether or not to begin a slideshow display over the results.

The *ROCS view* colors the reference molecule using the "Reference Color" (the default is green) as well as making that molecule *Locked*. The rest of the results are either overlayed in that same pane or displayed in their own individual pane overlayed on just the reference depending on whether or not Tiled mode was specified. An example of this can be seen in Figure 5.23(c).

As mentioned in Section 5.1.2, the *ROCS View* adds a custom pane to the Style Control to enable more specific interactions with the scene generated by *ROCS View* (see Figure 5.25).

The custom pane contains a single toggle button followed by a "Setup" button which relaunches the original *ROCS View* dialog (see Figure 5.21(c)). The single toggle button toggles the display of the reference molecule.

CHAPTER

SIX

2D Display

2D molecular depictions are provided using OpenEye's Ogham toolkit. A large number of options are available in the applications preferences to customize how the depictions are drawn. Depictions are drawn as black on a white background by default as seen in Figure 6.1, but other color schemes are available and can seen in Figure 6.2.

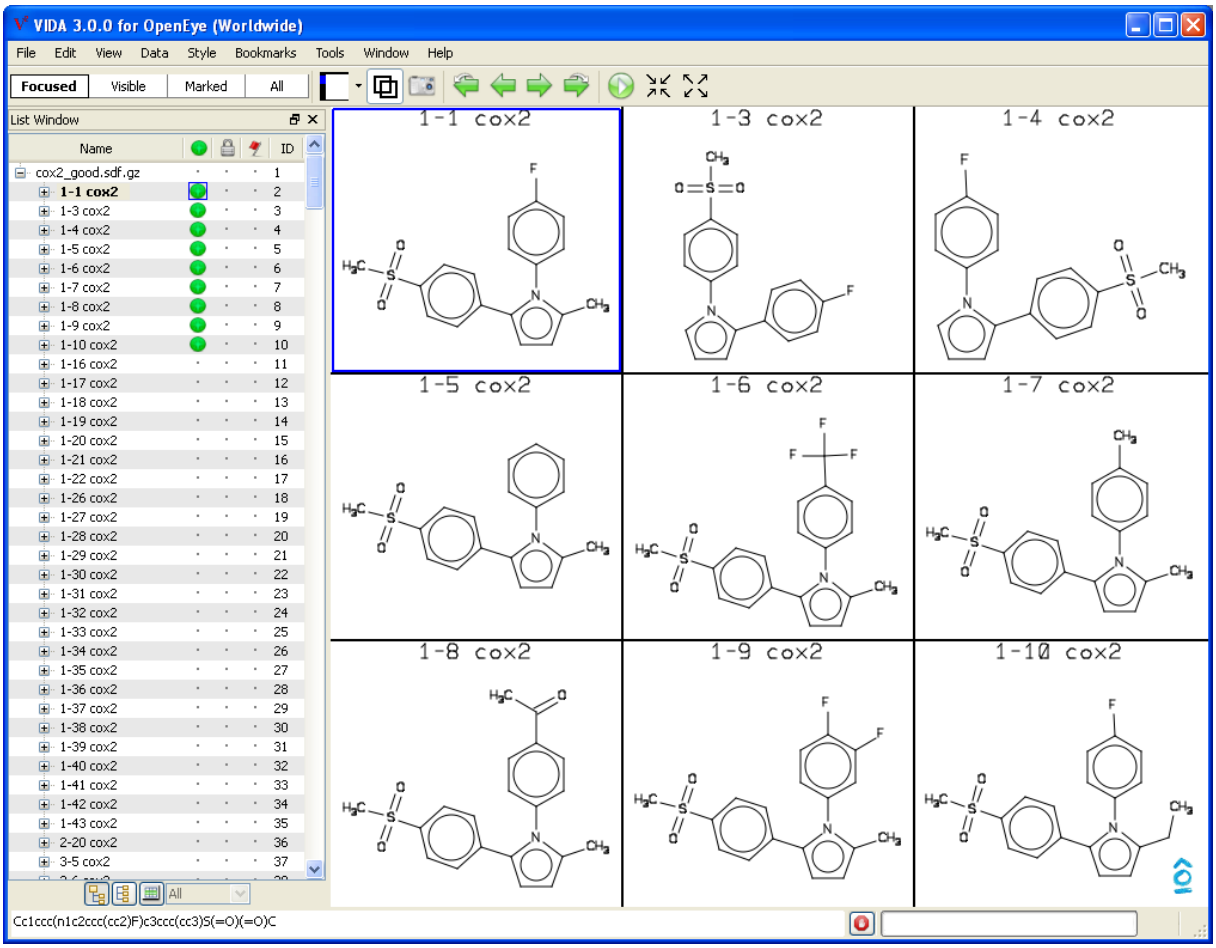
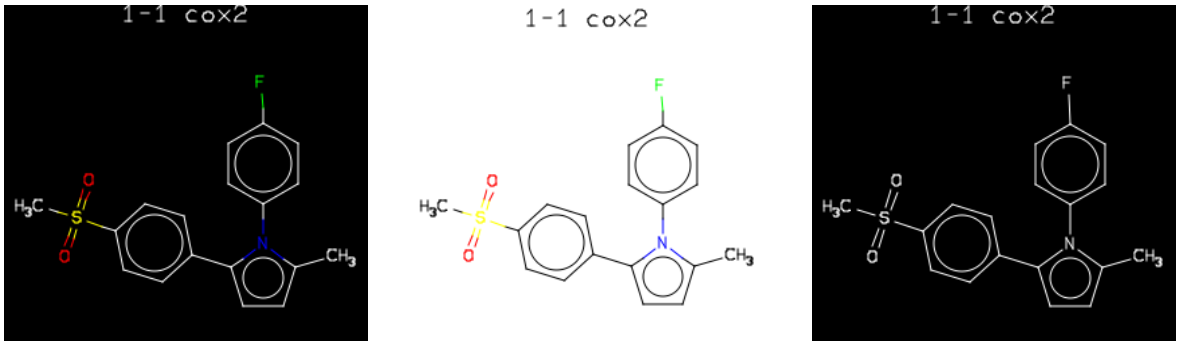


Figure 6.1: 2D Depiction



(a) Color on Black (b) Color on White (c) White on Black

Figure 6.2: Depiction color schemes

As might be expected, 2D depictions of this nature only make sense for small molecules (< 255 atoms in this case). For proteins, an amino acid sequence view is displayed instead (see Figure 6.3).

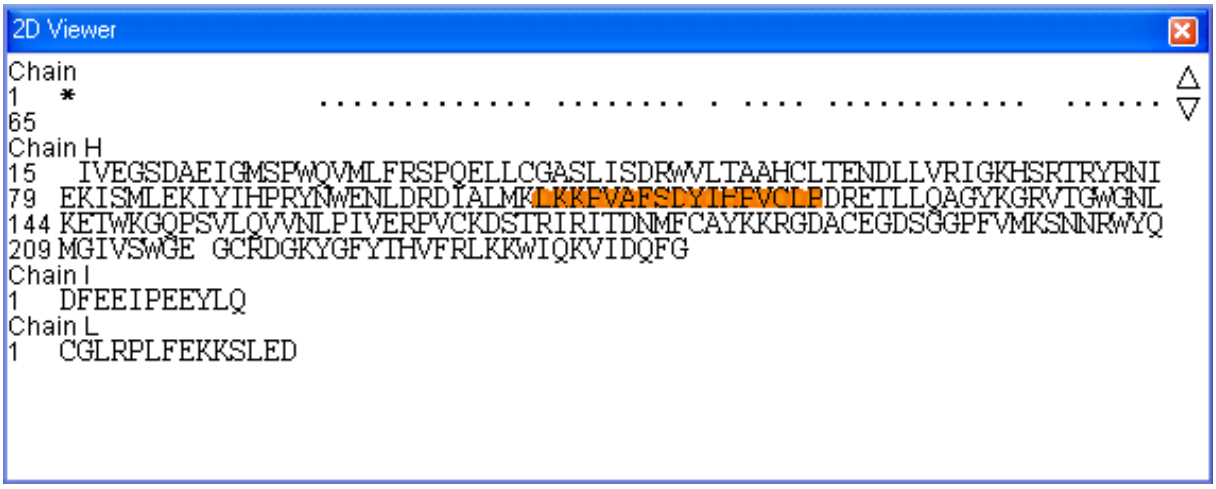


Figure 6.3: Amino Acid Sequence

6.1 Preview vs. Viewer

The 2D display comes in two flavors - the 2D Preview and the 2D Viewer. There are two differences between these two windows. First, the 2D Preview can only show a single molecule – the *Focused* one (see Section 3.1.1) – whereas the 2D Viewer can show several in a grid if the application is in Tiled mode. Secondly, the 2D Viewer can be set as the main window, but the 2D Preview cannot. The 2D Viewer is shown in Figure 6.1.

The 2D Preview is best suited for use as a small secondary display of the *Focused* molecule while otherwise browsing. The 2D Viewer often requires more screen real estate and is usually best suited as

the main window.

6.2 Interaction

2D selection behaves much like selection in the 3D display (see Section 5.1.1). Left clicking will select an individual atom, bond, or residue (if in sequence mode). When viewing a depiction, the right mouse button allows lasso style selection, as in the 3D display. When viewing a sequence, holding down the *Shift* key when selecting will allow selection of a continuous range of residues.

The current selection is reflected in the coloring of the 2D depiction. There may be slight variations between the depicted selection in 2D and 3D due to the difference in how hydrogens are drawn.

CHAPTER

SEVEN

Spreadsheet

The spreadsheet window is used to view, compute, and analyze data associated with molecules and can be seen in Figure 7.1. The spreadsheet window contains three initial spreadsheets by default: *Molecules*, *Proteins* and *Atoms*. The *Atoms* spreadsheet is slightly different in behavior from the other two and is described later.

The *Molecules* spreadsheet is populated with all currently loaded small molecules (fewer than 255 atoms). Conversely, the *Proteins* spreadsheet is populated all currenty loaded molecules that have more than 255 atoms.

The columns and their associated values are populated from SD data. SD data can be found in both SDF and OEB files. Additional spreadsheet columns can also be added by computing expressions (see Section 7.5) or by importing external data from a comma-seperated value (.csv) or tab-seperated value file (.txt) (see Section 7.9).

7.1 Basic Usage

Columns are divided into two basic types: data and property. Data columns are those found in SD data and/or imported files. Property columns include:

- Depiction - This column contains an image of 2D structure of the associated molecule.
- VIDA ID - This column contains the unique identifier assigned to that object by VIDA.
- VIDA Name - This column contains the title of the molecule as sepcified in the input file.
- Visible - This column contains an indicator (green dot) showing whether the molecule is *Visible* (see Section 3.1.2).
- Marked - This column contains an indicator (red flag) showing whether the molecule is *Marked* (see Section 3.1.4).
- Locked - This column contains an indicator (padlock) showing whether the molecule is *Locked* (see Section 3.1.3).

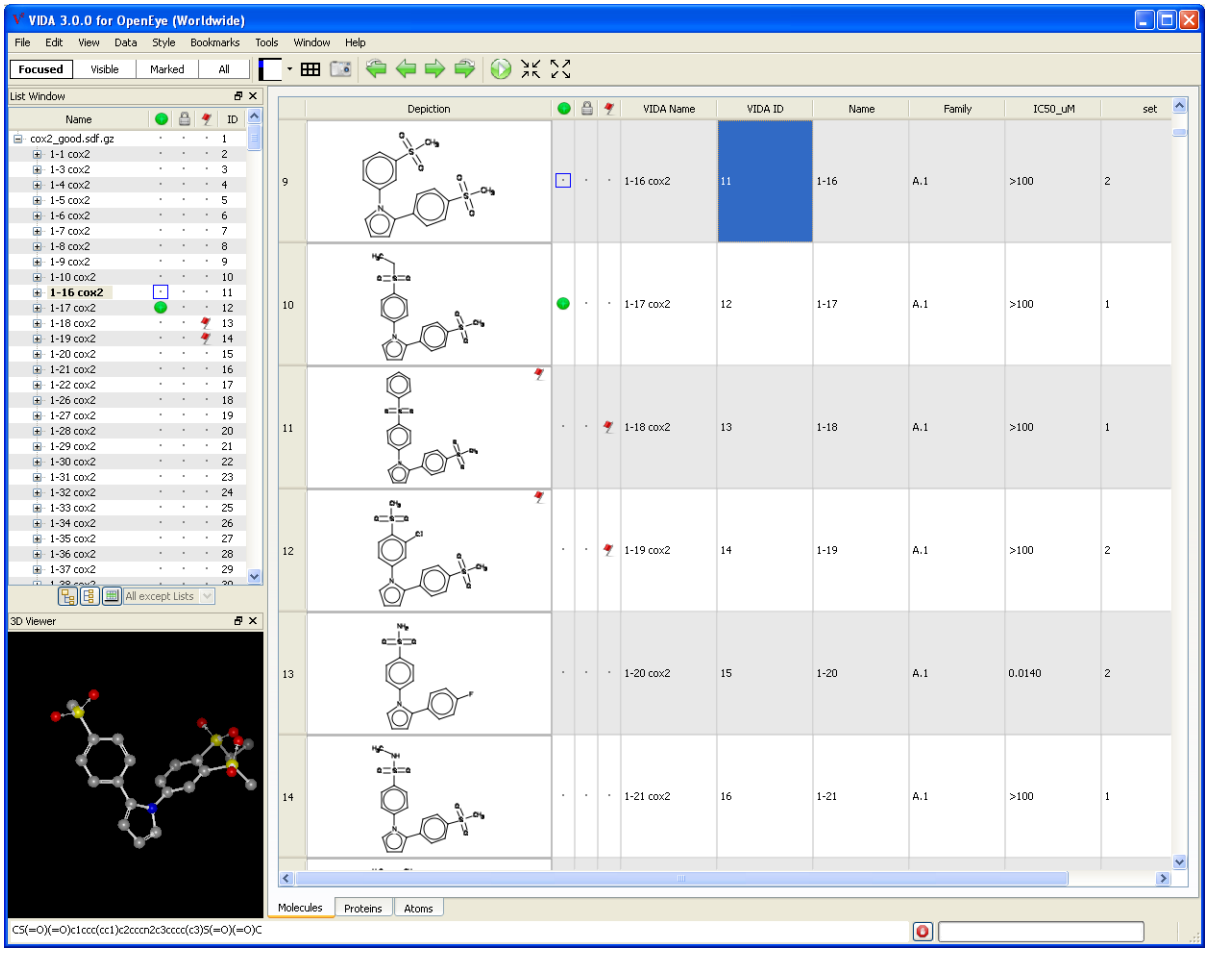


Figure 7.1: Spreadsheet window

Clicking on any data cell in the spreadsheet will cause the associated molecule to become the *Focused* molecule (see Section 3.1.1). Double-clicking on a cell in the spreadsheet enables modification of the data value in that particular cell. Some columns cannot be changed, such as the "VIDA ID" or "Depiction" columns.

Clicking on a cell and the holding the mouse button down while dragging selects a rectangular region in the spreadsheet. This region can be copied and pasted into other spreadsheets. The depiction column will be automatically converted to a SMILES string when feasible (molecule has fewer than 100 atoms). Similarly, clicking the mouse on a molecule's depiction and dragging the mouse to another application will transfer that image to the other application. This provides an easy way to transfer molecular images to other documents.

Clicking on a cell in the property columns *Visible*, *Marked* or *Locked* will toggle the respective property. To assign a property to a selected range of cells, simply select the range and then right click on the spreadsheet and select the appropriate action from the popup menu.

7.2 Sorting

A spreadsheet can be sorted by clicking on a column header. That spreadsheet will be sorted according to the data values in the associated column. If the data in the column to be sorted contains both numeric and non-numeric values, each set (numeric and non-numeric) will be sorted independently and then concatenated together.

Multiple columns can be sorted simultaneously by selecting the "Sort Spreadsheet..." option from the top level *Data* menu. This will launch a dialog that can be seen in Figure 7.2. This dialog can also be launched by right-clicking on any column header and selecting the "Sort Spreadsheet..." option from the generated popup menu.

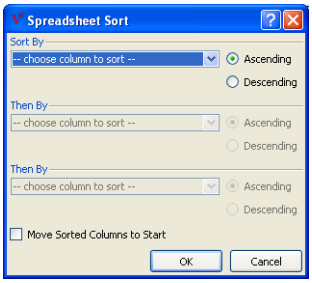


Figure 7.2: Sorting the spreadsheet

This sorting dialog allows for the selection of up to three columns. The spreadsheet will be sorted by the first column selected, with ties being broken by the remaining columns. The sorted columns can be moved to the front of the spreadsheet by selecting the "Move Sorted Columns to Start" option in the dialog.

The first time the spreadsheet is sorted, it may take longer than expected as the data values need to be retrieved from the molecule and any expressions must be calculated. The progress of the sort is indicated

in the progress bar at the bottom of the application. If desired, the sort can be aborted by clicking on the stop button to the left of the progress bar.

7.3 Displaying Data

The visibility of columns can be toggled by selecting the associated column name from the right-click popup menu generated by clicking on any column header. Data column names can be found in the "Column Visibility" submenu. Hiding the depiction column can allow for more rows to be displayed at the same time.

Certain molecular properties, such as molecular weight, can be added as new columns by selecting the desired option from the "Molecular Properties" submenu in the right-click popup menu. Some of these properties, such as Energy, might not be set on the molecule and as such the associated cell will contain an empty value.

See Section 7.5.2 for a complete listing of the molecule properties available for calculation.

7.4 Organizing Columns

Columns can be organized in the spreadsheet by selecting the "Organize Columns" option in the top level *Data* menu. This will launch a dialog which can be seen in figure 7.3.

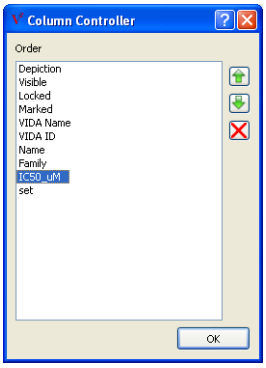
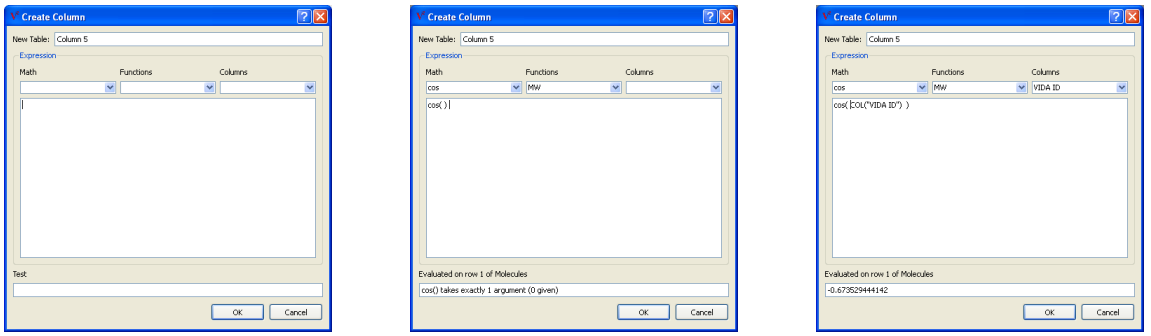


Figure 7.3: Spreadsheet organization dialog

The order of the columns can be controlled by selecting individual columns and then clicking on the up or down arrow buttons to change that columns position in the order. Furthermore, columns can be deleted by clicking on the button with the red X icon. Please be aware that deleting a column will delete the associated data from all currently loaded molecules and spreadsheets.

7.5 Creating New Columns



(a) Creating a new column

(b) Using a math function in the expression

(c) Referencing another column

Figure 7.4: Creating a new column from an expression

New columns can be added to the spreadsheet containing values that are based on or calculated from molecular properties or existing spreadsheet data. To create a new columns, select the "Create Column" option in the top level *Data* menu. This will launch a dialog to guide the process and can be seen in Figure 7.4.

The text entry field at the top contains the name for the new column. A new, empty and editable column can be easily be created by supplying a new column name and then clicking the "Create" button.

Beneath the name is a collection of three drop down boxes containing functions which can be used in building the expression to be evaluated when populating the column. The first drop down box contains a collection of available mathematical functions which can be used in the expression (see Section 7.5.1 for more details). The second drop down box contains a collection of available molecular properties that can be used in the expression (see Section 7.5.2 for more details). The last drop down box contains a list of the other already existing columns which can be used as part of the expression.

Any of the above described functions can be added to the expression simply by selecting them in the appropriate drop down box. If that function requires additional input (such as functions like *cos*), the cursor in the expression editor window will automatically be placed inside the function. Functions can be combined with standard math operators such as *, +, -, and . If a function requires additional input that is not provided, an error will be displayed in the Test section at the bottom of the dialog as seen in Figure 7.4(b). Frequently, the desired input is contained within another column which can be referenced by selecting the desired column from the third drop down box or simply by typing "COL('COLUMN NAME')" as seen in Figure 7.4(c).

Once the desired expression has been assembled, the column can be created and added to the spreadsheet by clicking on the "Create" button.

Name	Description
acos(x)	Return the arc cosine (measured in radians) of x.
asin(x)	Return the arc sine (measured in radians) of x.
atan(x)	Return the arc tangent (measured in radians) of x.
atan2(y,x)	Return the arc tangent (measured in radians) of y/x. Unlike atan(y/x), the signs of both x and y are considered.
ceil(x)	Return the ceiling of x as a float. This is the smallest integral value >= x.
cos(x)	Return the cosine of x (measured in radians).
cosh(x)	Return the hyperbolic cosine of x.
degrees(x)	Converts angle x from radians to degrees.
e	The numeric constant "e" or 2.7182818284590451.
exp(x)	Return e^x.
fabs(x)	Return the absolute value of the float x.
floor(x)	Return the floor of x as a float. This is the largest integral value <= x.
fmod (x,y)	Return the floating point remainder of x/y.
hypot(x,y)	Return the Euclidean distance: $\sqrt{x*x+y*y}$.
ldexp(x,y)	Returns $x * 2^i$.
log(x[,base])	The logarithm of x to the given base. If the base is not specified, returns ln(x).
log10(x)	The base 10 logarithm of x.
pi	The numeric constant PI or 3.1415926535897931.
pow(x,y)	Return x^y.
radians(x)	Converts angle x from degrees to radians.
sin(x)	Return the sine of x (measured in radians).
sinh(x)	Return the hyperbolic sine of x.
sqrt(x)	Return $\sqrt{x}$.
tan(x)	Return the tangent of x (measured in radians).
tanh(x)	Return the hyperbolic tangent of x.

Table 7.1: Available math functions

7.5.1 Math Functions

A large number of math functions are available for use in spreadsheet expressions as described in the previous section. The currently available math functions can be found in Table 7.1.

7.5.2 OEChem Functions

A large number of molecular properties are available for use in spreadsheet expressions as described above. The currently available molecular properties can be found in Table 7.2.

Name	Property
MW	Molecular weight.
Num Atoms	Number of atoms in the molecule.
Num Bonds	Number of bonds in the molecule.
Carbon-Hetero Ratio	Ratio of carbons to hetero atoms in the molecule. Returns 1 if there are no carbons in the molecule.
Energy	Molecular energy of the molecule as specified in the input file.
Actual Charge	Sum of the partial charges on all atoms.
Formal Charge	Sum of the formal charges on all atoms.
Halide Count	Number of halogen atoms in the molecule.
Num Carbons	Number of carbon atoms in the molecule.
Num Formal Charges	Number of atoms with a specified formal charge.
Num Heavy Atoms	Number of heavy atoms (non hydrogen) in the molecule.
Num Hetero Atoms	Number of hetero atoms in the molecule.
Num Hydrogens	Number of hydrogen atoms in the molecule.
Num Rigid Bonds	Number of rigid bonds in the molecule.
Nom Rotatable Bonds	Number of rotatable bonds in the molecule.
Num Chiral Atoms	Number of chiral atoms in the molecule.
Num Chiral Bonds	Number of chiral bonds in the molecule.

Table 7.2: Available molecular properties

7.6 Filtering

New spreadsheets may be generated from existing one by means of filtering. The mechanism to filter a spreadsheet is very similar to that of create a new column (see Section 7.5). To create a filtered spreadsheet, select the "Filter Spreadsheet..." option from the top level *Data* menu. This will launch a dialog which can be seen in Figure 7.5. This dialog will assist in the creation of a new filtered spreadsheet based on the currently viewed spreadsheet. Please note, that it is possible to create filtered spreadsheets on other filtered spreadsheets. The result of a filtering operation can be seen in Figure 7.6.

Filter expressions are very similar to column generation expressions (see section 7.5), except that the return value of a filter expression determines whether or not that row will be included in the new spreadsheet. Rows for which the expression returns either *False* or *0* will not be included.

7.7 Statistics

Statistics for each numeric column can be viewed in the spreadsheet window. The display of statistics is controlled by toggling the "Show Statistics" option in the top level *Data* menu. Toggling this option controls the display of statistics for just the currently viewed spreadsheet.

The following statistical information is computed for each numeric column:

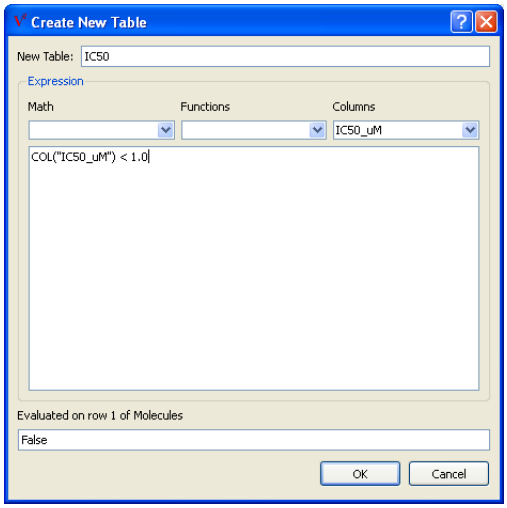


Figure 7.5: Filter dialog

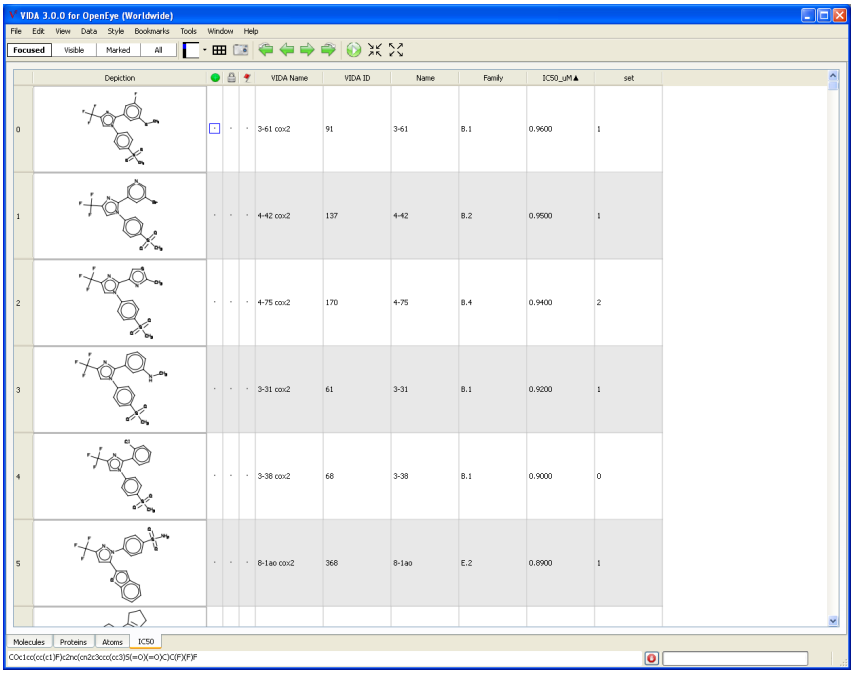


Figure 7.6: A filtered spreadsheet

- *min* - The minimum value of the column.
- *mean* - The maximum value of the column.
- *stddev* - The standard deviation of the column.
- *skewness* - The skewness of the column.
- *kurtosis* - The kurtosis of the column.

The statistics display is a dynamic view of the spreadsheet data and therefore when data is altered in the spreadsheet the statistics view is automatically updated. For large data sets, this may take some time and could cause editing the spreadsheet to be cumbersome. To avoid this problem, it is best to hide the statistics display until the editing process is complete.

7.8 Formatting Columns

There are a variety of different display styles for spreadsheet columns which can be edited on an individual basis. The customizable styles include color, font, significant digits, and cell height. The style for individual columns can be edited by selecting the "Format Columns..." option in the top level *Data* menu. This will launch the dialog seen in Figure 7.7.

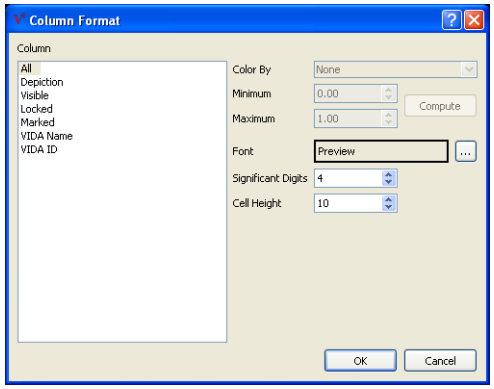


Figure 7.7: Spreadsheet format dialog

On the left hand side of the dialog is a list of all the currently available spreadsheet columns, including an *All* option. Any formatting changes made in this dialog will be applied to the selected columns in this list.

On the right hand side of the dialog is a collection of formatting and display options. At the top is a drop down menu of available coloring schemes which include:

- *Red to Blue*
- *Blue to Red*

- *Rainbow*
- *Rainbow Backwards*
- *Red Yellow Blue*
- *Grey to Green*

The selected coloring scheme will be applied to the column using the scale specified beneath the drop down menu determined by the minimum and maximum values. The default values are 0.0 and 1.0 respectively, but the current minimum and maximum values for the selected column can be computed by clicking on the "Compute" button to the right of the minimum and maximum fields.

7.9 Importing/Exporting

7.9.1 Importing

Spreadsheet data can be imported from external files in a variety of formats including comma-separated values (.csv) and tab-delimited values (.txt). To import a file, select the "Spreadsheet..." option in the *Import* submenu of the top level *File* menu. This will launch a dialog to assist in the import process which can be seen in Figure 7.8.

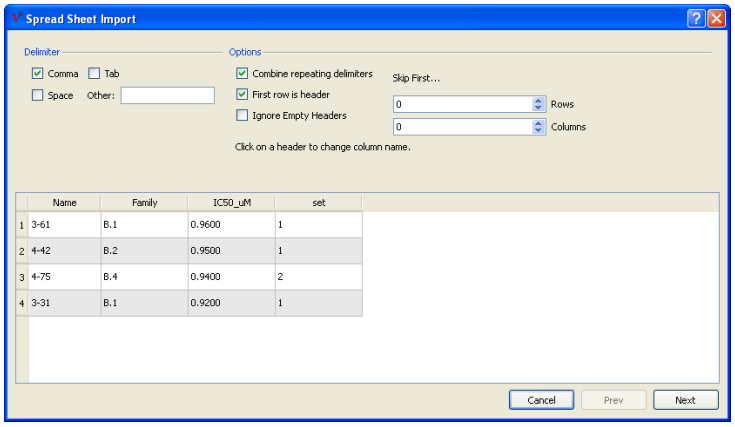
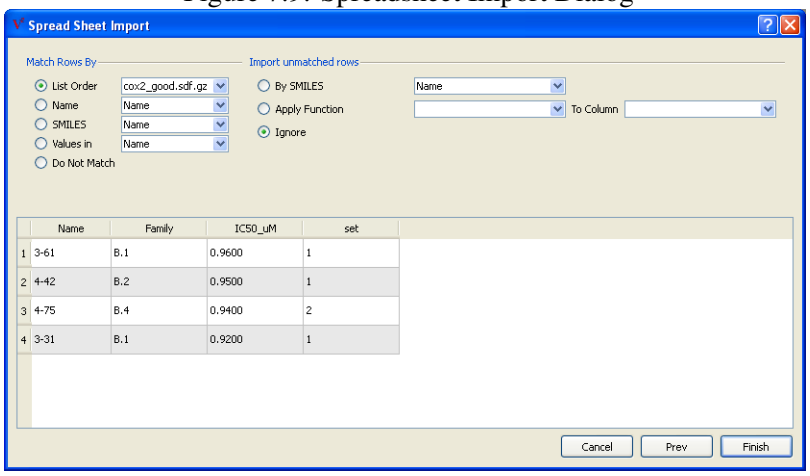


Figure 7.8: Spreadsheet import dialog

At the top of the dialog are a number of options to specify the format of the file to be imported and at the bottom is a quick preview of what VIDA thinks the first five rows are. The preview will update dynamically to match any changes made to the options at the top.

Once the preview of file to be imported is satisfactory, click on the *Next* button to specify how the rows are to be matched to currently loaded molecules and what to do with rows that do not match (see Figure 7.9).

Figure 7.9: Spreadsheet Import Dialog



There are a number of mechanisms available to match data rows to the appropriate molecules in VIDA:

- List Order - matches the order of rows in the spreadsheet with the order of molecules in the selected list.
- Name - matches the molecule's name with the name in the selected column of the imported data.
- SMILES - matches the molecule's canonical SMILES string with the SMILES string in the selected column of the imported data.
- Values In - matches the value in the selected column between the imported data and the value already in the spreadsheet.
- Do Not Match - assume that you are importing new molecules

Unmatched rows can also be imported into the spreadsheet by creating a new molecule in VIDA to associated with that row. The following options are available for unmatched rows:

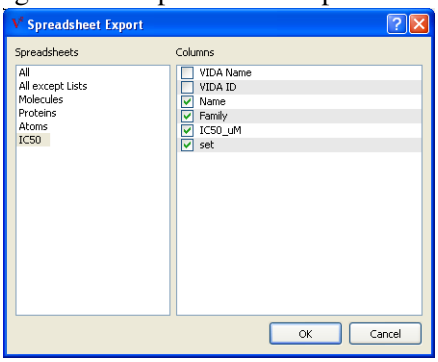
- By SMILES - use the SMILES string in the selected column of the imported data to create a new molecule.
- Apply Function - Apply a Python function to the data in the selected column of the imported data to create a new molecule. This can be particularly useful in cases where one of the columns contains a database ID which can be used to retrieve molecules.
- Ignore - Do not import unmatched data.

7.9.2 Exporting

Spreadsheets may be exported to an external file by selecting the "Spreadsheet..." option in the *Export* submenu of the top level *File* menu. Selecting this option will launch a dialog which allows for specifi-

cation of which columns to export (see Figure 7.10). Supported formats include either comma separated (.csv) or tab delimited (.txt).

Figure 7.10: Spreadsheet Export Dialog



CHAPTER

EIGHT

List Management

The organization of molecules into lists and the ability to easily browse through these lists is an extremely important and central concept to VIDA. As such a List Window is provided to aid in the navigation and organization of lists. Furthermore, there exist many mechanisms to create, modify, and delete lists which will be discussed in more detail later in this chapter.

8.1 Introduction

The List Window is one of the primary navigation and organization mechanisms in VIDA. It displays all of the currently loaded files and their contents (molecules, grids, surfaces, etc.) into hierarchical lists for easy browsing and organization.

The List Window can be seen in Figure 8.1 and contains multiple columns for controlling the various states and properties of the associated objects. The first column displays the name. The second column (with the green sphere icon at the top) is the *Visible* state column (see Section 3.1.2). The third column (with the padlock icon at the top) is the *Locked* state column (see Section 3.1.3). The fourth column (with the flag icon at the top) is the *Marked* state column (see Section 3.1.4). The fifth and last column contains the unique identifier assigned to that object.

8.2 Browsing

Every list in the List Window, as well as many of the other objects, can be expanded in the List Window to reveal further detail or the members contained within them. A small box with a "+" sign inside to the left of an object's name indicates that that object can be expanded. Molecules, for example, may be expanded to reveal conformers, molecular structure, or potentially other related children objects such as surfaces or grids. A single mouse click on the box will expand that object to reveal its contents. Double-clicking on the object name will turn on in-place renaming of that object.

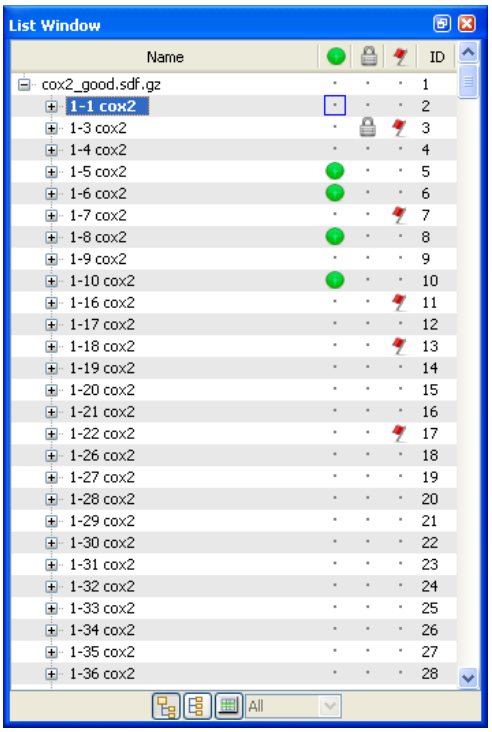


Figure 8.1: List window

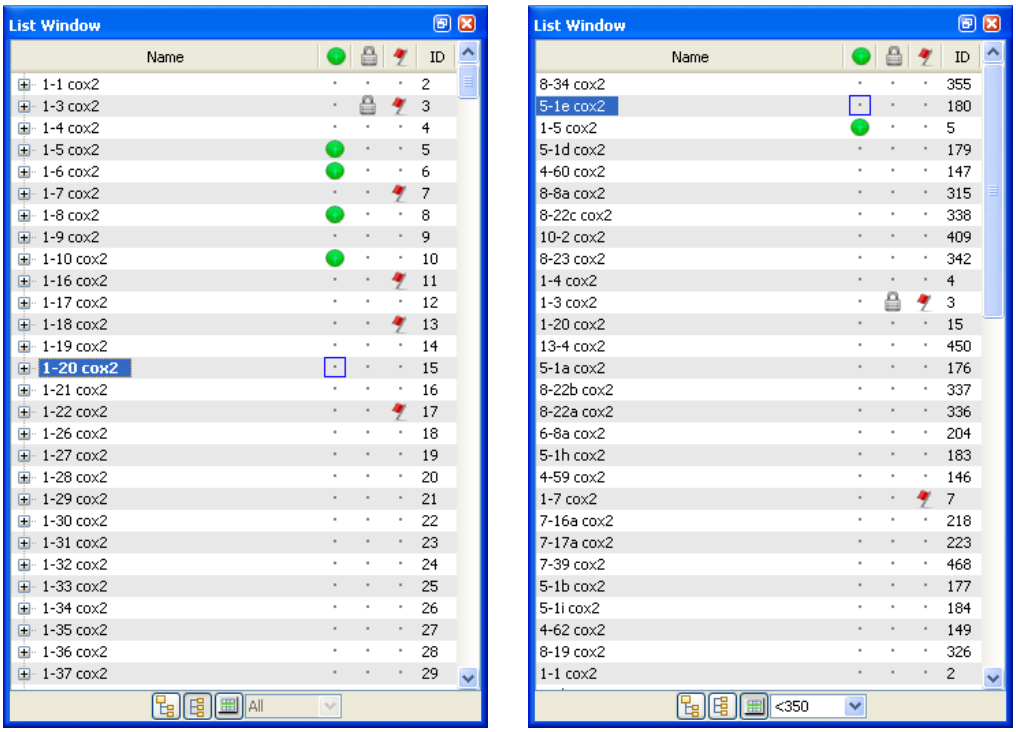
8.3 Views

The List Window provides three different views onto the loaded data. Which view is shown is controlled by the three buttons at the bottom of the List Window. The first view is a traditional hierarchical view, described above, and seen in Figure 8.1. The second view is a flat view (see Figure 8.2(a)) in which the list hierarchy has been removed, but all the contents are shown in the same original order. The third view is a spreadsheet view (see Figure 8.2(b)) which shows only those objects that appear within the associated spreadsheet (as specified by the drop down box next to its button). Furthermore, the order of the objects in this view parallels the current sorting order in the application spreadsheet. For more details about the spreadsheet, please see Chapter 7.

8.4 List Columns

Every object in the List Window may populate one of the several columns described in the introduction. Some columns are simply informative, such as the unique identifier, while others are functional as well as informative.

Clicking on an object's name will make that object the *Focused* object (see Section 3.1.1). In the List Window, the *Focused* object's name is displayed in bolded text and there will be a small blue box around the dot in the second column (the *Visible* column). An example of this can be seen in Figure 8.1, where



(a) Flat view (b) Spreadsheet view

Figure 8.2: Alternate views

the molecule “1-1 cox2” is *Focused*.

The second column is the *Visible* column. The *Visible* column has a green dot icon in its column header. This column displays and controls the *Visible* property for objects. If an object is *Visible* there will be a green dot identical to the one in the column header present in that object’s *Visible* column. If that object is also the *Focused* object, there will be a small blue box around the dot in that column as well.

If a given object has children, the visibility of its children can also be controlled in this column. Child objects have an additional layer of control regarding visibility in that they can be hidden, visible, or can mimic its parent’s visibility. If an object is set to mimic’s its parents visibility, an upright triangle is shown instead of the standard green dot (as can be seen in Figure 8.3). Clicking in this column cycles through these three states in the following order: hidden, mimic, visible. Please note that molecular structure (atoms, residues, chains, etc) are not considered children objects and as such always mimic the parent molecule’s visibility if they are marked visible with a green dot.

The third column is the *Locked* column. The *Locked* column has a small, grey padlock icon in its header. If an object is *Locked* there will be a grey padlock identical to the one in the column header present in that object’s *Locked* column.

The fourth column is the *Marked* column. The *Marked* column has a red check mark icon in its header. If an object is *Marked* there will be a red check mark identical to the one in the column header present in that object’s *Marked* column.

The fifth and final column is the *ID* column. An object’s ID is simply a unique identifier that can be

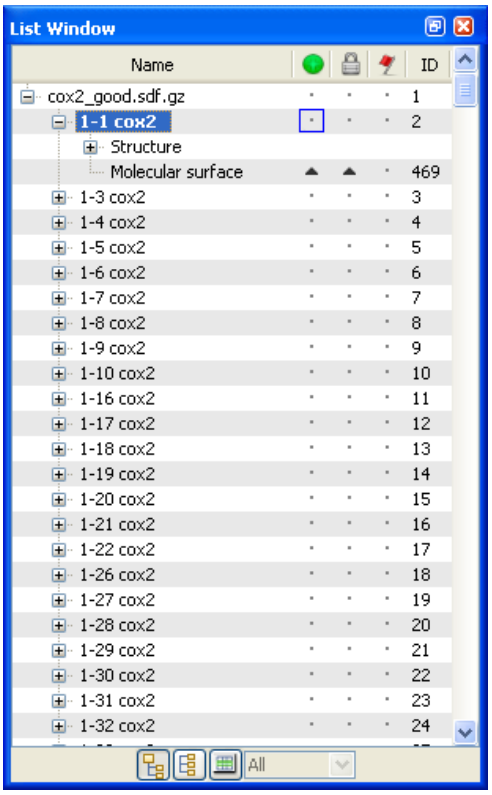


Figure 8.3: Child object mimicing parent

useful to identify specific objects when running various commands. Each top level object has its own unique ID, but child objects like conformers and atoms are all a part of their parent molecules's ID.

Columns can be shown or hidden as desired by right-clicking in the header of the List Window. These changes will be remembered in the Preferences (see Chapter 11) and thus persist between sessions.

The *Visible*, *Locked*, and *Marked* properties can be toggled by clicking on their associated columns in the List Window. If multiple objects are highlighted simultaneously and the click occurs in a column associated with the highlighted region, that change will apply to all of the highlighted objects. Compare Figure 8.4 to Figure 8.1. In Figure 8.4, the highlighted objects have been made *Visible* by clicking in that column. The *Focused* object has changed as a result as well.

8.5 List Manipulation

The List Window operates on the basic concept that every potentially viewable object is placed inside a list. As a result, the root items in the List Window are always lists. These lists can be combined, subsetted, and manipulated into new lists. The properties of all the objects in a list can be set together by clicking in the columns of their containing list. Finally, objects can easily be removed from and inserted into lists.

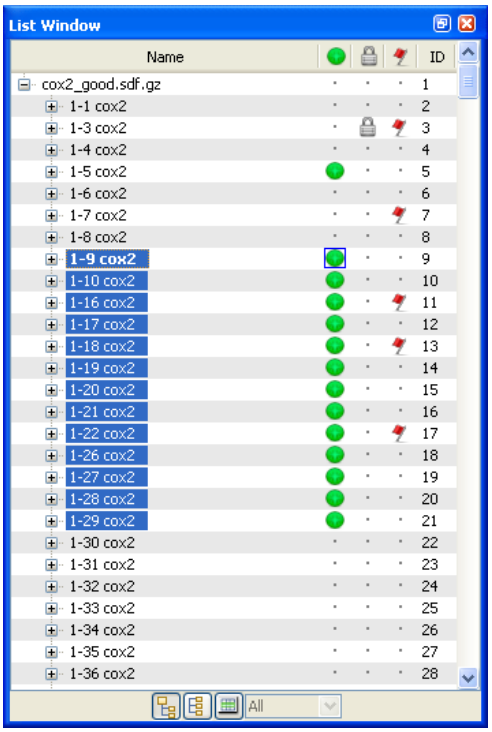


Figure 8.4: Changing multiple objects simultaneously

8.5.1 Creating Lists

New lists can be created in several ways. The simplest way is the "New List" submenu in the *File* menu. A number of list creation mechanisms are available and include using the AND of two or more lists (intersection), the OR of two or more lists (union), the XOR of two lists (union - intersection), or from the currently *Marked* set of objects.

The AND list contains only those objects which are in all of the source lists. The OR list contains all of the objects contained in its individual source lists. The XOR list contains the objects which are in only one of the source lists, but not both. Finally, the *Marked* list creates a new list which contains all of the currently *Marked* objects.

The second way to create a new list is by right-clicking on an individual list and selecting the "Subset" submenu from the popup menu. This menu contains four entries – "By Marked", "By Query", "By Not Marked" and "By Not Query". The "By Marked" option creates a new list containing all of the *Marked* objects from within the list. The "By Not Marked" option creates a list with all of the objects from the list which are not *Marked*. "By Query" generates a dialog which prompts for a substructure query that if the objects match, those objects will be placed in the new list. The "By Not Query" behaves the same way, except that it puts the objects that do match the query into the new list.

The last way to create a new list is from the "Find" menu entry in the *Edit* menu. This mechanism searches all of the currently loaded molecules by either Amino Acid Sequence, SMARTS, a regular expression against the molecule title, or by a Query molecule (as a substructure search). After clicking the "Search" button, the matching results will be displayed on the right-hand side of the dialog. The

Figure 8.5: List navigation buttons

entire set or just specific individual results can be selected within the dialog to populate the new list.

8.5.2 Deleting from Lists

As previously mentioned, objects can also be removed from lists. There are two ways to remove an object from a list. The first mechanism is to delete the object (by selecting the "Delete" option from the right-click menu). Deleting an object will remove it from ALL of the lists to which it belongs. Choosing the "Remove from List" option from the same right-click menu will remove the object from the specified list only, leaving it perfectly intact in any other lists that it might belong to. When the last object is removed from a list, the list itself will be removed.

8.6 Toolbar

This List Window installs a collection of five navigations button in the application toolbar which can be see in Figure 8.5. These buttons provide an additional mechanism to browse through lists and their contents in VIDA. In addition, these buttons are aware of the current list view as described in Section 8.3 and as such operate on that view.

The first four buttons in the set browse to the beginning, to the previous object, to the next object, and to the last object respectively. The final button with the "play" icon initiates a slideshow which steps forward to the next object after a certain period of time (this can be set in the application preferences). The slideshow can be paused by clicking on the same button again (which should have changed to have a "pause" icon once the slideshow started). The slideshow will automatically stop when it reaches the end of the objects in the current view of the List Window.

CHAPTER

NINE

Process Management

The Process Manager allows the user to start, monitor, terminate, and load results from processes external to VIDA. As part of this functionality, it is possible to write out currently loaded molecules (or grids) to files which can be used as input to the external processes. Furthermore, it is possible to create "pipes" which chain together the input and output of individual processes into a larger process.

The Process Manager contains two tabbed windows called "Processes" and "Status" respectively as can be seen in Figures 9.1 and 9.2. The "Processes" tab contains the list of available processes and the functionality to create and edit them. The "Status" tab displays all of the currently active and recently completed processes and information about their state. It also displays the commandline output generated by each of the processes.

Beneath the list of processes in the "Processes" tab is a collection of four buttons. The button on the left with the play icon starts the currently selected process. The next button with the plus icon clears the editor on the right hand side to allow for creation of a new process. The next button with the link icon changes the editor on the right hand side to allow for creation of a new pipe (see Figure 9.6). The last button with the delete icon deletes the current selected set of processes from the list of available processes.

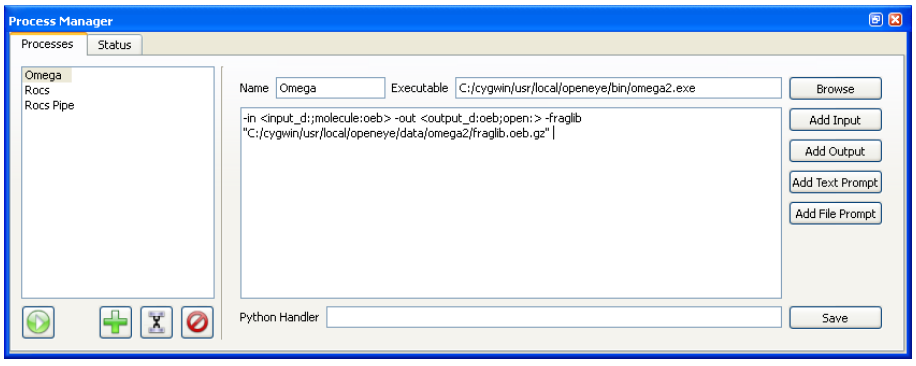


Figure 9.1: Process Manager

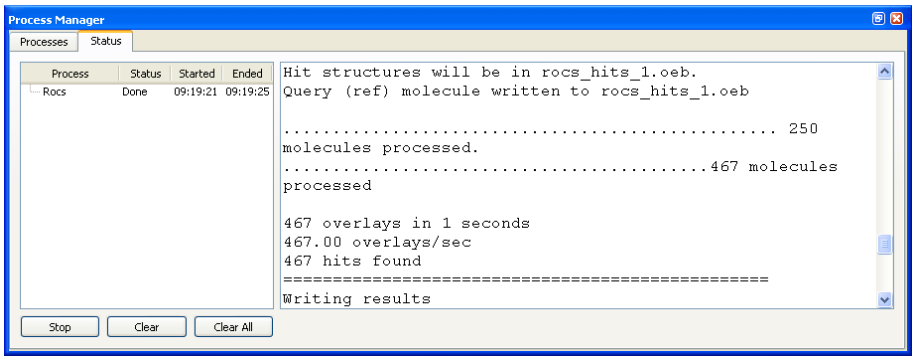


Figure 9.2: Output generated by completed ROCS process

9.1 Processes

When creating a new process or editing a current one, there are a number of available options. At the top of the editor there is a field for the process name (which will be displayed in the list of available processes seen on the left as described above) as well as a field for the actual location of the application to be run. The "Browse" button to the right of this field can be used to find the location of the desired application.

Beneath this is a large text entry box which allows for specification of any desired command line arguments to the application. It is not necessary to enter the application name here.

For most applications, at least one filename will have to be specified as a command line argument. This can be done multiple ways. If the file to be specified is always going to be located in the same location, the filename can be entered directly into the text entry box as a command line argument. If the location of the file is not likely to be constant, a prompt can be specified as opposed to an actual filename. To add a prompt, click on the "Add File Prompt" button to the right of text entry box. More details on adding prompts can be found in Section 9.1.3. If the desired file will need to be created each time, it is possible to create the desired file at start-time by writing out selected molecules (or grids) directly from VIDA. To use this feature, click on the "Add Input" button to the right of the text entry box. At start-time, the user will be prompted to select the desired molecules (or grids), which will then be saved to a file which will be passed as a command line argument to the process. For more details on adding an input file see Section 9.1.1.

In addition to being able to specify specific input files, it is possible to specify an output file which can potentially be read back into VIDA when the process is completed. As discussed above, it is possible to a specify specific filename in the command line or to use a prompt, but if it desired to load the resulting file, the "Add Output" button should be clicked to specify the file. For more details on adding an output file see Section 9.1.2.

Besides being able to specify filenames, it is often desirable to be able to specify other variable arguments at start-time. To do this, click on the "Add Text Prompt" button which will add a prompt argument to the command line arguments. At start-time the user will be asked to specify the desired text to be passed to the process. For more details on text prompts see Section 9.1.3.

Lastly, at the bottom of the Process Editor is a "Python Handler" field which allows for specification

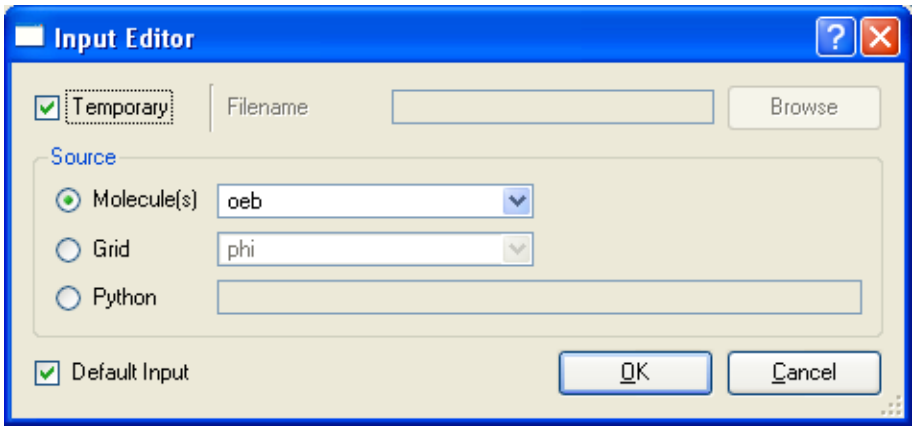


Figure 9.3: Process Input Editor

of a Python function to be run when the process is completed. This allows for handling of output not normally handled by VIDA or for output which is not specified on the command line.

9.1.1 Adding Input

The Input Editor allows the user to specify a mechanism by which an input file is provided to the process to be run. At the top of the editor are two mutually exclusive options, the first is the "Temporary" check box which indicates that the file to be used will be a temporary file which will be removed after the process has completed. The other option is a field which allows specification of a specific filename to be used. In both cases, the user has a choice as to how this file will be populated: saving molecules, saving grids, or running a specified Python function which will handle the population of the file. If molecules or grids are chosen, a file format to save these in must be specified in the pull-down box next to the selected option.

If a Python function is specified to populate the input file, the desired filename can be obtained by using the keyword "@FILE@" in the function specification. This keyword will be substituted with the real filename at start-time before the Python function is called.

Finally, there is a checkbox at the bottom of the dialog to specify whether or not this is a default input. The default input entry is used when constructing pipes and will be discussed in Section 9.2.

9.1.2 Adding Output

The Output Editor allows the user to specify a mechanism by which an output file of the process to be run is handled by VIDA. At the top of the dialog are two mutually exclusive options, the first is the "Temporary" check box which indicates that the output file is a temporary file and will be removed after it has been handled by VIDA. The other option is a field which allows specification of a specific filename to be used for the output file. If this option is selected, the file will not be deleted after it is handled.

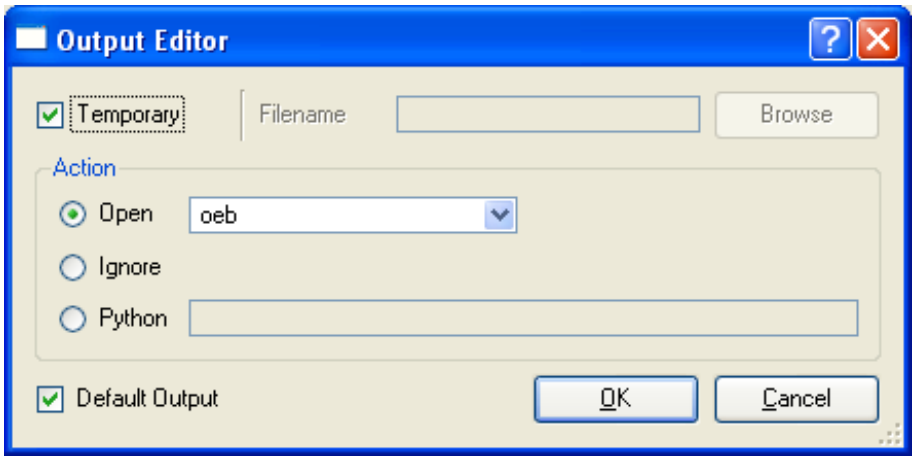


Figure 9.4: Process Output Editor

Once the above has been specified, the user can specify how VIDA should handle the file when the process is finished. There are three possible options, the first is to open the file, the second is to ignore the file, and the last is to call a Python function to handle the resulting file. If it is desired to open the file, the file format that the process should use to generate the output file must be specified so that VIDA will know how to properly load the results.

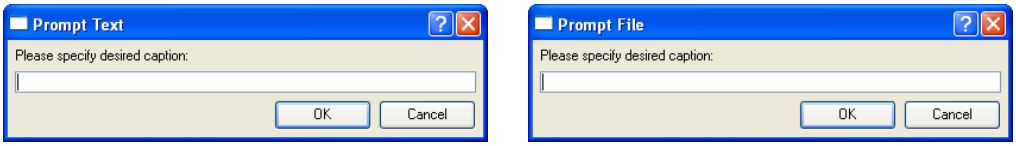
If a Python function is specified to handle the output file, the actual filename can be obtained by using the keyword "@FILE@" in the function specification. This keyword will be substituted with the real filename at process completion time before the Python function is called.

Finally, there is a checkbox at the bottom to specify whether or not this is a default output. The default output entry is used when constructing pipes and will be discussed in Section 9.2.

9.1.3 Adding Prompts

As described in Section there are two types of prompts that are available for specification on the command line: a text prompt and a file prompt.

The Text Prompt Editor (see Figure 9.5(a)) allows the user to add a text prompt to the process as well as to specify what the caption of the prompt will be. Specification of an informative caption is helpful in allowing the user to know what information is being requested.



(a) Text Prompt Editor (b) File Prompt Editor

Figure 9.5: Prompt Editors

The File Prompt Editor (see Figure 9.5(b)) allows the user to add a file prompt to the process as well as to specify what the caption of the resulting prompt will be. Specification of an informative caption is helpful in allowing the user to know what specific file or type of is being requested.

The result of both of these prompts are passed as command line arguments to the process in the order in which they appear in the main text entry box in the Process Editor (see Figure 9.1).

9.2 Pipes

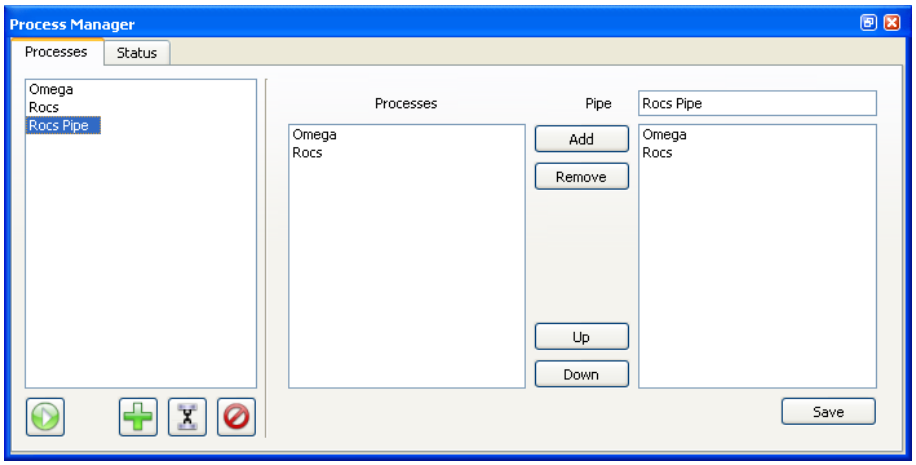


Figure 9.6: Pipe Editor

Once a number of processes have been created, they can be "piped" together using the Pipe Editor (see Figure 9.6). On the left-hand side of the editor is a list of all the currently defined processes. On the right is an ordered list of processes to be run and in which order to run (top to bottom).

When chaining processes together into a pipe, it is important that they have default input and output fields specified (see Sections 9.1.1 and 9.1.2). The default output of a given process is passed into the default input of the following process. When the final process is completed, the user will be notified and the final output file will be handled based on its specification in the associated process.

CHAPTER

TEN

Scripting

VIDA is in part built on top of the Python scripting language, which is an interactive, interpreted, strongly and dynamically typed language. More information about Python can be obtained from the Python website located at *http://www.python.org/*. The website contains many useful tutorials and other related information.

All user interaction with the application goes through a Python layer even if the user is not aware of it. For instance, every button in the application is actually associated with a Python command that is sent to the internal interpreter when that button is pressed. The same integration is true of all the menu items and other GUI controls in the application. In fact, the journal file written by the application to save its history is itself a Python script which can be run to regenerate the state of the application.

The journal file (`journal.py`) is written to a user-specific local directory on the computer currently running the application. On Microsoft Windows 2000 and XP, the journal can be found in `C:\Documents and Settings\USERNAME\Application Data\OpenEye\VIDA\3.0`. On Microsoft Windows Vista, the journal can be found in `C:\Documents and Settings\USERNAME\AppData\Local\OpenEye\VIDA\3.0`. On all other platforms, the journal can be found in `~USERNAME/.OpenEye/VIDA/3.0`. If the application were to unexpectedly exit, the journal file corresponding to that run will be saved with a separate file containing a time stamp of when the application exited. These journal files are particulary useful for debugging unexpected application behavior and are of extreme benefit to the application developers when tracking down user-reported bugs. The inclusion of journal files (if possible) is greatly encouraged when submitting bug reports.

The complete details of the Python interface is fully described in a seperate API manual which documents every possible command that can be run within the application.

10.1 Scripting Window

As mentioned above Python is an interactive language and as such a scripting window is provided as an interface to Python. The scripting window can be seen in Figure 10.1.

The scripting window consists of two components – input and output. The large top component of the window is the output area which displays all of the output from commands entered in the bottom input

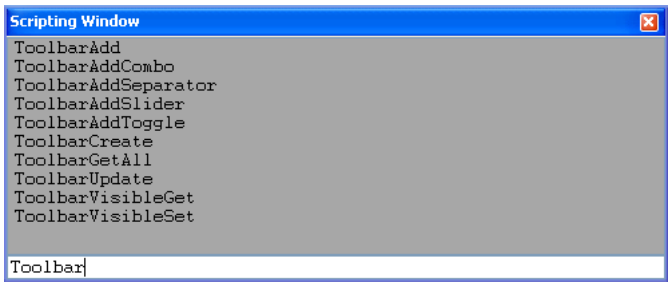


Figure 10.1: Scripting window with tab completion of Python commands

area. It also displays any warnings or errors generated during the run of the application. The input component is the single line text entry area at the bottom of the window. On the input line, the user can directly type in Python commands to be run by VIDA.

The scripting input line is intelligent. Hitting the *Tab* key at the beginning of a line will indent the entry line to the previous line's identation if there was any indentation at all. If there was no indentation, the line will indent five spaces. However, hitting the *Tab* key after typing a portion of a command will cause the scripting line to expand out that command as far as possible without introducing ambiguity. For example, hitting the *Tab* key after typing "Tool" will expand out to "Toolbar." At this point, hitting the *Tab* key again will result in a listing (in the output display above) of all the commands that begin with "Toolbar" (see Figure 10.1). Finally, hitting the *Tab* key when a command is fully expanded will display the API documentation associated with that command in the output display.

10.2 OEChemLite

`oechemlite` is a Python module that ships with VIDA which enables access to the currently loaded molecules via the scripting interface. `oechemlite` is distinguished from OpenEye's PyOEChem module in that it `oechemlite` provides only the PyOEChem functions that are not capable of making modifications to the molecules. Future versions of VIDA will enable the import of the full PyOEChem module which will allow modifications, provided that a separate license for the OEChem Python wrappers has been obtained.

Using `oechemlite`, a molecule is accessed by calling the `ScriptableObjectGet` function which takes an ID as a parameter and returns the associated molecule. For more details on the associated API please see the PyOEChem documentation; however, please remember that only the "const" functions (those that do not make changes to their objects) are available in `oechemlite`.

As most scripting routines in VIDA take unique keys as parameters, a `GetKey` function has been added to every object which returns the associated object's unique key identifier. In addition, the following methods have also been added to all `oechemlite` objects for convenience:

1. IsActive()
2. IsMarked()

3. IsLocked()
4. IsSelected()
5. IsVisible()
6. SetLocked(value = True)
7. SetMarked(value = True)
8. SetSelected(value = True)
9. SetVisible(value = True)

Two example scripts which use `oechemlite` appear below:

```
def SelectCarbonAtoms(idOrKey):
  """Select all carbon atoms from the given id or key
  if an id is used, only carbons from the first conformer are selected"""
  PushIgnoreHint()
  try:
    mol = ScriptableObjectGet(idOrKey)
    mol.SetVisible()

    for atom in mol.GetAtoms():
      if atom.IsCarbon():
        atom.SetSelected()

  finally:
    # the try..finally will always call
    # the popignorehint even if there is an error
    # otherwise, the application might not work correctly
    PopIgnoreHint()

def CreateSphereMonitorAtomsByDistance(idOrKey, x, y, z, distance):
  """idOrKey, x, y, z, distance -> generate sphere monitors for all atoms
  belonging to the specified id or key with distance from coordinate x,y,z
  When using an ID only atoms from the first conformer will be examined"""
  distanceSquared = distance * distance

  PushIgnoreHint(True)
  try:
    mol = ScriptableObjectGet(idOrKey)
    mol.SetVisible()

    color = OEColor(255, 255, 0, 100)
    for atom in mol.GetAtoms():
      x1,y1,z1 = mol.GetCoords(atom)
      dx = x-x1
      dy = y-y1
      dz = z-z1
```

```
      if dx*dx + dy*dy + dz*dz < distanceSquared:
        CreateSphereMonitor(atom.GetKey(), "Within Distance", color)

  finally:
    # the try..finally will always call
    # the popignorehint even if there is an error
    # otherwise, the application might not work correctly
    PopIgnoreHint()
```

Please note that it is not necessary to `import oechemlite` to access this functionality, that is done automatically at startup.

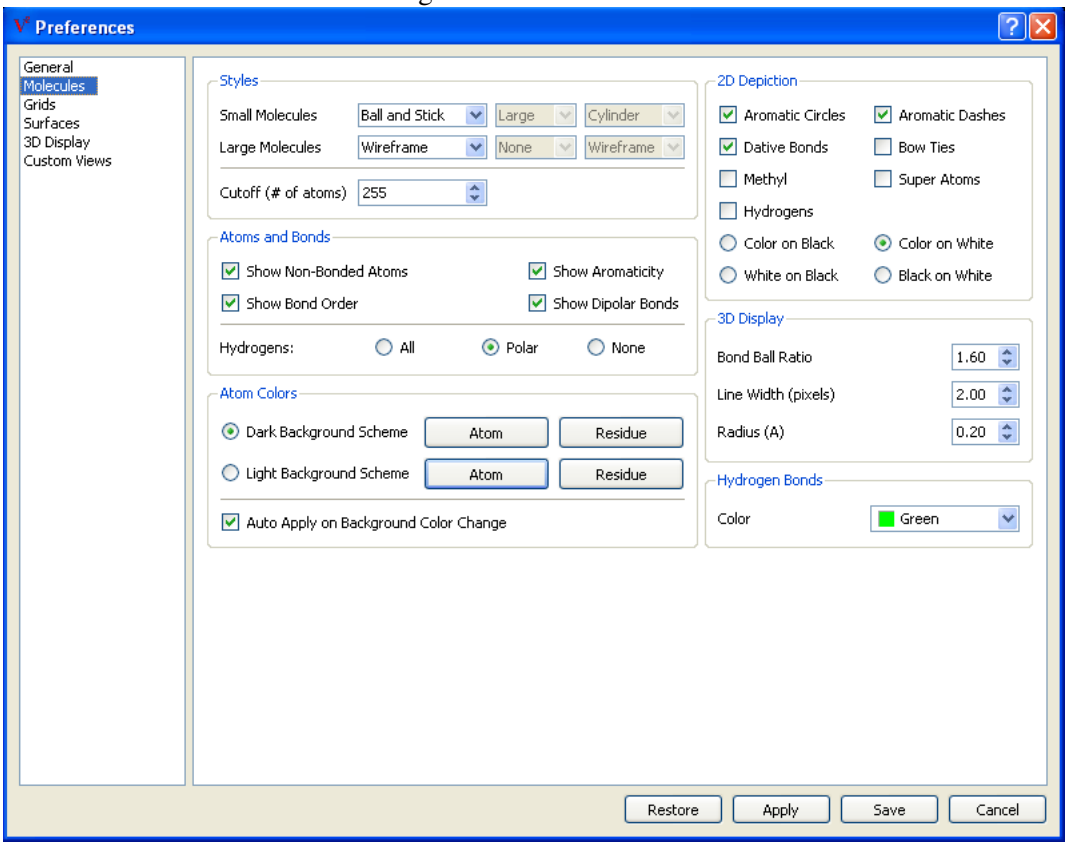
CHAPTER

ELEVEN

Preferences

Every user has his or her own individual preferences with regards to how molecules, grids, and surfaces should look and how applications should behave. For this reason, a Preferences dialog is available which allows customization of the application to the user's preference. An snapshot of the Preferences dialog can be seen in Figure 11.1.

Figure 11.1: Preferences



On the left-hand side of the dialog is column containing preference categories. These categories include: *General*, *Molecules*, *Grids*, *Surfaces*, *3D Display*, and *Custom Views*. Clicking on any of these categories will update the right-hand side of the widget to display the options corresponding to the selected category.

There are four buttons at the bottom of the dialog. Clicking on the *Restore* button will restore the current preferences to the default ones. Clicking on the *Apply* button will apply the current preferences to this run only and will not be automatically saved when VIDA is exited. Clicking on the *Save* button will save the current preferences for this run and for future runs of VIDA. Clicking on the *Cancel* button will close this dialog and will not apply any of the changed preferences.

Preferences are stored in a binary file (`preferences.oeb`) in a user-specific local directory on the computer currently running the application. On Microsoft Windows 2000 and XP, the preferences file can be found in `C:\Documents and Settings\USERNAME\Application Data\OpenEye\VIDA\3.0`. On Microsoft Windows Vista, the preferences can be found in `C:\Documents and Settings\USERNAME\AppData\Local\OpenEye\VIDA\3.0`. On all other platforms, the preferences can be found in `~USERNAME/.OpenEye/VIDA/3.0`.

While the preference file shares the same file extension as the OpenEye's binary database file, it cannot be read into VIDA using the "Open" menu item in the *File* menu. The preferences file is loaded automatically when the applications starts and is saved back to disk when the application exits. Deleting this file is equivalent to clicking on the *Restore* button in the dialog box.

There is also a file in this same directory which called VIDA.ini which contains machine specific settings like the list of recents files, prefered layouts, and hardware stereo options. Deleting this file will restore these settings to the defaults as well.

This directory can be opened from within VIDA by selecting the "Open User Directory" menu item in the *Help* menu.

CHAPTER

TWELVE

Examples

12.1 Data

VIDA ships with a directory of example data for the user to experiment with in the event that other input files are not immediately available. Included in this directory are:

- `cox2_good.sdf.gz` - This is a large set of Cox-2 inhibitors which contains associated molecular data which makes this an excellent example file for use in the Spreadsheet.
- `MMFF94.mol2.gz` - This is another large set of small molecules but it contains a significantly greater amount of diversity among the molecules than the previous file. This file is an excellent example file for browsing through a large data set as well as for exploring the many different ways that shape and chemistry may be visualized with VIDA.
- `1ll5.grd` - This is a grid file containing electron density data corresponding to crystallography data of the ligand structure found in the following file. It provides an excellent example of VIDA's grid visualization capabilities and also works well to provide a context for the associated ligand file.
- `1ll5.lig.pdb.gz` - This small molecule is a beta-lactamase inhibitor positioned in its crystallographically determined binding location. This file combined with the actual beta-lactamase protein found in the following file provide an excellent example of how VIDA can be used to visualize small molecule-protein interactions. When visualizing this molecule, be sure to observe the three excellent hydrogen bonds it makes with the protein.
- `1ll5.nolig.pdb.gz` - This file contains the crystallographically determined structure of a beta-lactamase protein. Its bound inhibitor can be found in the previous file. This file is an excellent example of the many ways that proteins can be visualized in VIDA. Be sure to observe its many internal hydrogen bonds, particularly in the active site.
- `1ll5.omega.oeb.gz` - This file contains a large number of conformations of the beta-lactamase ligand (`1ll5.lig.pdb.gz`) generated with OpenEye's conformer generation program *Omega*. This file is an excellent example of VIDA's ability to handle multi-conformer molecules as well as its ability to view *Omega* output. It is important to note that this file is

not in the same coordinate space as the three previous files and as such should not be expected to appear in the same location on screen with the above files.

12.2 Scripts

VIDA ships with directory of example scripts to highlight the different aspects of VIDA's scripting capability. Included in this directory are:

- `catalyst.py` - This script provides a very basic example of how one can extend VIDA to read and visualize pharmacophores from a Catalyst® output file. Loading this script will create a new menu item called "Catalyst..." in the "Open Special" menu. Selecting this item will prompt the user to locate an appropriate input file which will be read and then visualized.

- `omega.py` - This script provides an example of how to extend VIDA such that it can launch OpenEye's conformer generation program *Omega* on data currently loaded in VIDA. The script will need to be customized by the end-user to specify the actual installation location of *Omega* on that system. Loading this script will create a new toolbar button with an omega symbol icon. Clicking on this button will start an external *Omega* process with the currently *Focused* molecule as its input. The results will be loaded into VIDA when the process finishes. This script also modifies the "Open" command such that it looks to see if the molecules being read in contain 3D coordinates or not. If no 3D coordinates are detected, it will prompt the user to see if it should generate 3D coordinates for these molecules using *Omega*.

- `pdbopen.py` - This script creates a new menu option in the *File* menu called "Open From PDB" which allows VIDA to open molecule files directly from the online Protein Data Bank. Simply enter the PDB ID and if VIDA is able to locate that ID in the database, it will download the file and load it into VIDA.

- `rocs.py` - This script provides an example of how to extend VIDA such that it can launch OpenEye's 3D shape overlay program *Rocs* on data currently loaded in VIDA. The script will need to be customized by the end-user to specify the actual installation location of *Rocs* on that system. Loading this script will create two new toolbar buttons, one with an "R" icon and one with an "R*" icon. Clicking on the "R" icon will prompt the user specify a set of molecules currently loaded in VIDA to use as the database to be searched. The query structure will be the currently *Focused* molecule. Clicking on the "'R" icon will prompt the user to specify a file on disk to be used as the database while the currently *Focused* molecule will be used as the query. When the process is finished, the results will be loaded automatically into VIDA.

CHAPTER

THIRTEEN

Help

VIDA provides a dynamic help utility for browsing as well as searching the provided application documentation. The help system provides access to both the user manual as well as the scripting API documentation. The help utility can be started by selecting the "Dynamic Help" option in the top-level *Help* menu.

The help utility behaves very similarly to a web browser and as such should be familiar to most people. In addition, the documentation is organized and indexed for quick and easy access. The documentation can also be searched for individual words or phrases. Bookmarks can easily be created as well to enable fast access in the future to frequently viewed pieces of documentation.

In addition to the dynamic help utility, documentation is also provided in PDF and HTML formats which can also be accessed via the top-level *Help* menu.

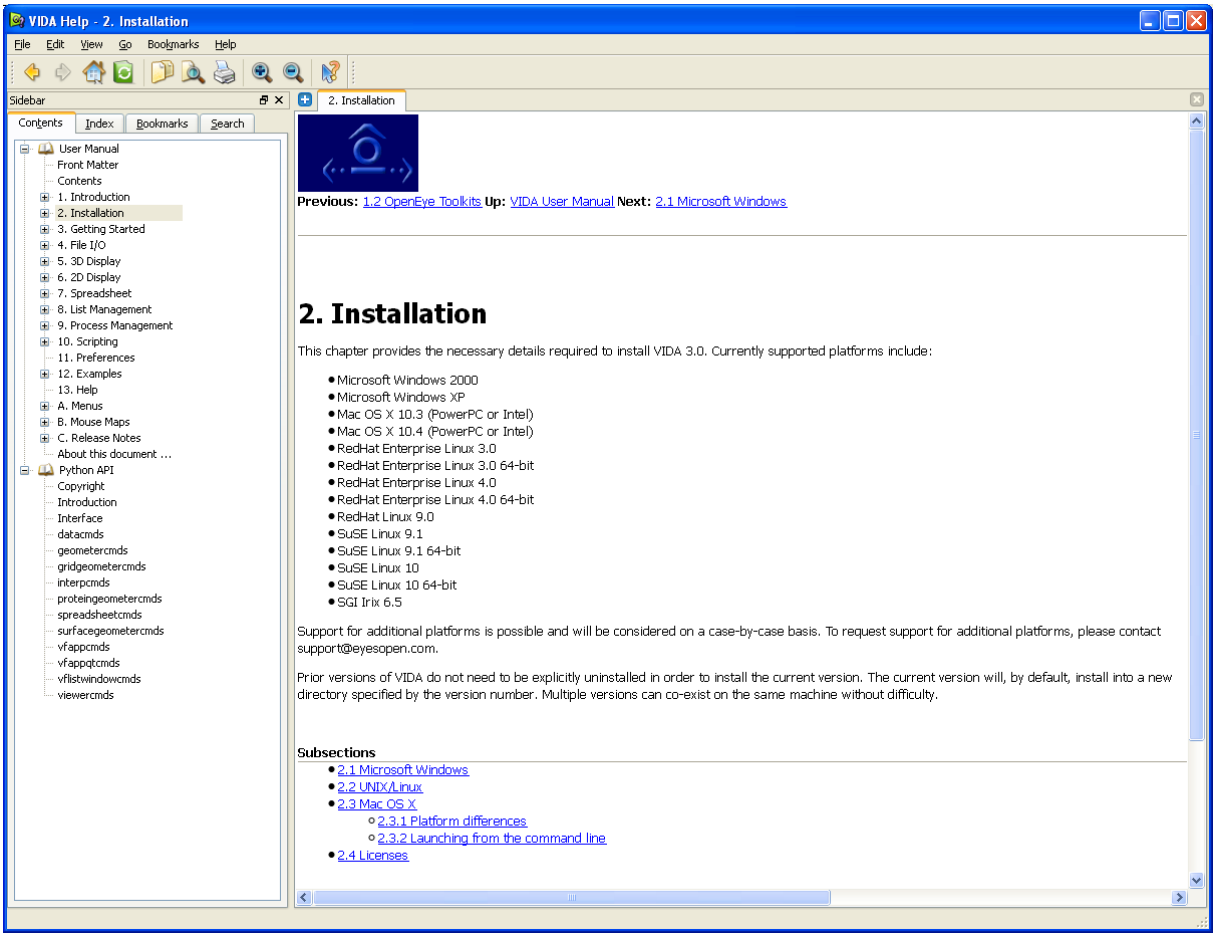


Figure 13.1: Dynamic Help

APPENDIX

A

Menus

This appendix contains a complete listing and detailed information of all the menu options available in VIDA. Each section also contains a listing of the Python code that is run when that option is selected.

A.1 File

A.1.1 Open

```
oname = ''
try:
  oname = PromptFilenames([], 'molecule,grid,surface,python')
except:
  pass

if (oname != ''):
  Open(oname)
```

A.1.2 Open State

```
OpenState(PromptFilename('','Open','state'))
```

A.1.3 Open Special

Script

```
Open(PromptFilename('','Open','python'))
```

A.1.4 Clear

```
DeleteAll()
```

A.1.5 Save

```
sname = ''
ids   = []

try:
  ids = PromptIDs()
  if (len(ids)>0):
    sname = PromptSaveFilename(ids)
except:
  pass

if (sname != '') & (len(ids) > 0):
  Save(sname, ids)
```

A.1.6 Save State

```
results = []
filters = 'OpenEye State Version 2 ( *.oes.gz ) ;;'      \
          'OpenEye Mini State Version 2 ( *.oes.gz ) ;;' \
          'OpenEye State Version 1 ( *.oes.gz ) '

try:
  results = PromptSaveFilename(filters)
except:
  pass

if (len(results) == 2):
  SaveStateFilter(results[0], results[1])
```

A.1.7 New List

From AND

```
ListNewAnd(PromptIDs(['Lists'], 0))
```

From OR

```
ListNewOr(PromptIDs(['Lists'], 0))
```

From XOR

```
ListNewXor(PromptIDs(['Lists'], 0))
```

From Marked

```
ListNewMarked()
```

A.1.8 New Molecule

From Merge

All three options under this menu (*Scoped*, *Marked*, and *Visible*) execute the following code.

```
id = MoleculeMergeScoped(SCOPE, PromptString('Name:'))
if id:
  Active(id)
```

The SCOPE parameter is assigned a value according to which option was selected (SelectedScope, MarkedScope, and VisibleScope respectively).

From Split

Components

```
ids = MoleculeNewSubsetScoped(0,ActiveThenVisibleScope,1)
if ids:
  Active(ids[0])
```

Selected

```
ids = MoleculeNewSubsetScoped(0,ActiveThenVisibleScope,SelectedScope,1)
if ids:
  Active(ids[0])
```

Marked

```
ids = MoleculeNewSubsetScoped(0,ActiveThenVisibleScope,MarkedScope,1)
if ids:
  Active(ids[0])
```

A.1.9 Import

Spreadsheet

```
eval( SpreadsheetPromptImport(PromptFilename('', 'Open', 'data')) or 'None' )
```

A.1.10 Export

Script History

```
AppScriptSave(PromptFilename('journal.py', 'Save', 'python'))
```

Screenshot

```
MainWindowScreenshotPrompt()
```

Spreadsheet

```
eval( SpreadsheetPromptExport(PromptFilename('', 'Save', 'data')) or 'None' )
```

POV-Ray

```
ViewerExportPOVRAY(PromptFilename('', 'Save', ''))
```

A.1.11 Recents

All of the options in this menu call the following command.

```
Open(FILENAME)
```

The FILENAME parameter is assigned to the filename specified in the option that was selected under this menu.

A.1.12 Exit

```
Quit()
```

A.2 Edit

A.2.1 Undo

```
Undo()
```

A.2.2 Undo History

A.2.3 Redo

```
Redo()
```

A.2.4 Redo History

A.2.5 Copy

```
Copy()
```

A.2.6 Copy Image

```
MainWindowScreenshot()
```

A.2.7 Paste

```
PasteMolecules()
```

A.2.8 Select

All Visible

```
SelectAllMolecules()
```

Query

```
SelectByQuery(PromptQuery(), False, VisibleScope)
```

Invert

```
SelectInvert()
```

None

```
ClearSelection()
```

A.2.9 Mark

All Visible

```
MarkObjectsByScope(VisibleScope)
```

Query

```
SelectByQuery(PromptQuery(), True)
```

None

```
ClearMarked()
```

A.2.10 Hide

All

```
ClearVisible()
```

Selected

```
OEPyHide(SelectedScope, True)
```

Marked

```
OEPyHide(MarkedScope, True)
```

Not Selected

```
OEPyHide(SelectedScope, False)
```

Not Marked

```
OEPyHide(MarkedScope, False)
```

None

```
OEPyHideNone()
```

A.2.11 Find

```
FindInRepository()
```

A.2.12 Find on Disk

```
FindOnDisk()
```

A.2.13 Preferences

```
PreferencesEdit()
```

A.3 View

A.3.1 Fullscreen

```
if AppShowGet() == 'fullscreen':
  AppShowSet('normal')
else:
  AppShowSet('fullscreen')
```

A.4 Data

A.4.1 Sort Spreadsheet

```
eval( SpreadsheetPromptSort() or 'None' )
```

A.4.2 Filter Spreadsheet

```
eval( SpreadsheetPromptFilter() or 'None' )
```

A.4.3 Create Column

```
eval( SpreadsheetPromptColumnExpression() or 'None' )
```

A.4.4 Format Columns

```
for x in SpreadsheetPromptFormat()[:-1].split(';'):
  eval(x or 'None')
```

A.4.5 Organize Columns

```
SpreadsheetColumnController()
```

A.4.6 Show Statistics

```
SpreadsheetShowStatsSet(SpreadsheetCurrent(), not SpreadsheetShowStatsGet(SpreadsheetCurrent()))
```

A.5 Style

A.5.1 Color

Palette

```
ColorSetScoped(PromptColor())
```

Unique

```
ColorUniqueScoped()
```

Restore

```
MoleculeColorsResetScoped()
```

Molecules

All but two of the options (*Carbon* and *Reference* in this menu call the following function.

```
MoleculeColorByScoped(SCHEME)
```

The SCHEME parameter is assigned based on which menu option was selected according to the table below.

MENU OPTION	SCHEME
Amino	‘amino’
BFactor	‘bfactor’
Chain	‘chain’
CPK	‘cpk’
CPKNew	‘cpknew’
Element	‘element’
Formal Charge	‘formal charge’
Group	‘group’
Partial Charge	‘partial charge’
Residue	‘residue’
Shapely	‘shapely’

The *Carbon* menu option calls the following function.

```
AtomColorSetScoped(6, PromptColor())
```

The *Reference* menu option calls the following function.

Reference

```
AtomColorReferenceScoped()
```

Surfaces

All of the options in this menu call the following function.

```
SurfaceColorByScoped(SCHEME, PARAMS)
```

Both the SCHEME and PARAMS parameters are assigned based on which option was selected according to the table below.

MENU OPTION	SCHEME	PARAMS
Atom Color	‘atom’	‘’
Concavity	‘concavity’	‘’
Curvature	‘curvature’	‘’
Distance	‘distance’	‘’
Electrostatic	‘electrostatics’	‘’
Grid	‘grid’	str(PromptID([’g’, False))
Hydrogen Bond Potential	‘hydrogen bonds’	‘’
Hydrophobicity / Charifson	‘hydrophobicity’	‘charifson’
Hydrophobicity / Eisenberg	‘hydrophobicity’	‘eisenberg’
Hydrophobicity / Kyte-Dolittle	‘hydrophobicity’	‘kyte-dolittle’
Hydrophobicity / White-Octanol	‘hydrophobicity’	‘white-octanol’
Surface Potential	‘potential’	‘’

Transparency

All of the options in this menu call the following function.

```
SurfaceTransparencySetScopedScoped(N)
```

The N parameter is assigned to a value between 0 and 255 based on which option was selected. A value of 255 corresponds to 0 percent transparency and a value of 0 corresponds with 100 percent transparency. The specific values used in the menu are: 255, 230, 204, 179, 153, 128, 102, 77, 61, 26, 0.

A.5.2 Selection

Select All

```
SelectAllMolecules()
```

Select By Query

```
SelectByQuery(PromptQuery())
```

Select Inside Range

```
SelectWithin(PromptFloat('Select inside range (Angstroms)'), True)
```

Select Outside Range

```
SelectWithout(PromptFloat('Select outside range (Angstroms)'), True)
```

Crop Selected

```
SurfaceCropScribedScoped()
```

Crop Unselected

```
SurfaceCropUnscribedScoped()
```

Invert Selection

```
SelectInvert()
```

A.5.3 Display

Molecules

All of the options in this menu call the following function.

```
MoleculeStyleSetScoped(STYLE)
```

The STYLE parameter is assigned based on which option was selected according to the table below.

MENU OPTION	STYLE
Wireframe	'wireframe'
Stick	'stick'
Ball and Stick	'ball and stick'
CPK	'cpk'
Stars	'stars'
Hidden	'hidden'

Hydrogens

All of the options in this menu call the following function.

```
AtomHydrogenStyleSetScoped(STYLE)
```

The STYLE parameter is assigned based on which option was selected according to the table below.

MENU OPTION	STYLE
All	2
Polar	1
None	0

Non-Bonded Atoms

Both of the options in this menu call the following function with the HIDE parameter assigned based on which option was selected (*Show* assigns HIDE to False and *Hide* assigns HIDE to True).

```
BondHideNonbondedSet(HIDE)
```

C-Alpha Trace

Both of the options in this menu call the following function with the DRAW parameter assigned based on which option was selected (*Show* assigns DRAW to True and *Hide* assigns DRAW to False).

```
DrawCATracesSetScoped(DRAW)
```

Ribbons

Both of the options in this menu call the following function with the DRAW parameter assigned based on which option was selected (*Show* assigns DRAW to True and *Hide* assigns DRAW to False).

```
DrawRibbonsSetScoped(DRAW)
```

Atom Labels

The *Atom Labels* submenu provides a variety of labeling of options. The "Custom..." option launches a dialog which allows the user to build up more complex and customized labels. This dialog is launched using the following Python command:

```
ViewerLabelDialog('Atoms')
```

All of the other options in the submenu all call the following Python command but with different string arguments:

```
AtomLabelSetScoped('')
```

The arguments used can be found in the table below:

Name	'%n'
Index	'%i'
Element	'%e'
Isotope	'%iso'
Type	'%t'
Integer Type	'%it'
Implicit H Count	'%h'
Map Idx	'%m'
Partial Charge	'%p'
Formal Charge	'%f'
Residue Info	'%r'
Residue Info (CA)	'%car'
B-Factor	'%cab'
Degree	'%d'
Is Chiral	'%c'
Chirality	'%chi'
Hybridization	'%hy'
Has Stereo Specified	'%sts'
Is In Ring	'%rng'
OEChem Idx	'%oei'
None	''

Bond Labels

The *Bond Labels* submenu provides a variety of labeling of options. The "Custom..." option launches a

dialog which allows the user to build up more complex and customized labels. This dialog is launched using the following Python command:

```
ViewerLabelDialog('Bonds')
```

All of the other options in the submenu all call the following Python command but with different string arguments:

```
BondLabelSetScoped('')
```

The arguments used can be found in the table below:

Name	'%n'
Index	'%i'
Type	'%t'
Integer Type	'%it'
Order	'%o'
Length	'%l'
Is Chiral	'%c'
Chirality	'%chi'
Has Stereo Specified	'%sts'
OEChem Idx	'%oei'
None	''

Electrostatic Potential

Both of the options in this menu call the following function with the DRAW parameter assigned based on which option was selected (*Show* assigns DRAW to True and *Hide* assigns DRAW to False).

```
ShowESGridScoped(DRAW)
```

Molecular Surface

Both of the options in this menu call the following function with the DRAW parameter assigned based on which option was selected (*Show* assigns DRAW to True and *Hide* assigns DRAW to False).

Show

```
ShowSurfaceScoped('molecular', DRAW)
```

Accessible Surface

Both of the options in this menu call the following function with the DRAW parameter assigned based on which option was selected (*Show* assigns DRAW to True and *Hide* assigns DRAW to False).

```
ShowSurfaceScoped('accessible', DRAW)
```

Hydrogen Bonds

Add Targets

```
HBondAddTargetsScoped()
```

Remove Targets

```
HBondRemoveTargetsScoped()
```

External Hydrogen Bonds

```
HBondShowExternalSet(not HBondShowExternalGet())
```

Internal Hydrogen Bonds

```
HBondShowInternalSet(not HBondShowInternalGet())
```

Grids

Solid

```
ContourDrawAsSurfaceSetScoped(True)
```

Mesh

```
ContourDrawAsSurfaceSetScoped(False)
```

Cloud

```
ContourDrawStyleSetScoped('cloud')
```

Grid Corners

```
GridShowCornersSet(not GridShowCornersGet())
```

Electrostatic

```
GridTypeSetScoped(1)
```

ET

```
GridTypeSetScoped(5)
```

FRED

```
GridTypeSetScoped(2)
```

Generic

```
GridTypeSetScoped(0)
```

Difference Map

```
GridTypeSetScoped(4)
```

Regular Map

```
GridTypeSetScoped(3)
```

Add Contour

```
ContourTypedAddScoped()
```

Delete Contour

```
ContourTypedRemoveScoped()
```

Surfaces

All of the options in this menu except one (*Restore*) call the following function.

```
SurfaceStyleSetScoped(STYLE)
```

The STYLE parameter is assigned based on which option was selected according to the table below.

MENU OPTION	STYLE
Solid	'solid'
Mesh	'mesh'
Points	'points'

The *Restore* option calls the following funcion.

```
SurfaceRestoreScoped()
```

A.5.4 Graphics

Depth Cue

```
ViewerDepthcueSet(not ViewerDepthcueGet())
```

Slabbing

```
ViewerSlabEnableSet(not ViewerSlabEnableGet())
```

Mirror Slabs

```
ViewerMirrorSlabsSet(not ViewerMirrorSlabsGet())
```

Slab Level

All of the options in this menu call the following function.

```
ViewerSlabEnableSet(True)
ViewerSlabFarSet(50+N/2)
```

The N parameter is assigned to a value between 10 and 80 corresponding to the desired percent of the scene to be contained within the slabbing planes.

Render Quality

All of the options in the menu call the following function.

```
ViewerLODSet(LOD)
```

The LOD parameter is assigned based on which option was selected according to the table below.

MENU OPTION	**LOD**
Fastest	1
Fast	2
Good	3
Better	4
Best	5

Stereo

Off

```
ViewerStereoEnableSet(False)
```

Splitscreen

```
ViewerStereoEnableSet(True)
ViewerStereoHardwareSet(False)
```

Hardware

```
ViewerStereoEnableSet(True)
ViewerStereoHardwareSet(True)
```

Angle

```
ViewerStereoAngleSet(PromptFloat('Stereo Angle'))
```

Separation

```
ViewerStereoSeparationSet(PromptFloat('Stereo Separation'))
```

Cross-Eyed

```
ViewerStereoCrossEyedSet(not ViewerStereoCrossEyedGet())
```

A.5.5 Symmetry

Edit Cell

```
XRayEditCrystalParams(ActiveID())
```

Show Unit Cell

```
DrawUnitCellSet(OEInternal.MenuVal)
```

Show Symmetry

```
DrawSymmetrySet(OEInternal.MenuVal)
```

Realize Symmetry In Cell

```
DoSymmetryRealize(True)
```

Realize Symmetry

```
DoSymmetryRealize(False)
```

Symmetry Radius

```
SymmetryRadiusSet(PromptFloat('Symmetry radius',SymmetryRadiusGet()))
```

Symmetry Operators

No Symmetry Color

```
SymmetryColorHelper(0,OEInternal.MenuVal)
```

Single Symmetry Color

```
SymmetryColorHelper(1,OEInternal.MenuVal)
```

Color By Operator

```
SymmetryColorHelper(2,OEInternal.MenuVal)
```

Unique Color

```
SymmetryColorHelper(3,OEInternal.MenuVal)
```

A.6 Bookmarks

A.6.1 Add

```
BookmarkSave(PromptString('Bookmark name', 'Bookmark ' + str(1 + len(Bookmarks()))))
```

A.6.2 Organize

```
exec(BookmarkOrganizeDialog())
```

A.6.3 Animated

```
BookmarksSetAnimated(not BookmarksGetAnimated())
```

A.7 Tools

A.7.1 FRED View

```
CustomViewFRED()
```

A.7.2 EON View

```
CustomViewEON()
```

A.7.3 ROCS View

```
CustomViewROCS()
```

A.8 Window

A.8.1 Main Window

2D Viewer

```
if OEInternal.MenuVal: AppMainWindowSet('2D Viewer')
```

3D Viewer

```
if OEInternal.MenuVal: AppMainWindowSet('3D Viewer')
```

Spreadsheet

```
if OEInternal.MenuVal: AppMainWindowSet('Spreadsheet')
```

A.8.2 2D Preview

```
WindowVisibleSet('2D Preview')
```

A.8.3 2D Viewer

```
WindowVisibleSet('2D Viewer')
```

A.8.4 3D Viewer

```
WindowVisibleSet('3D Viewer')
```

A.8.5 List Window

```
WindowVisibleSet('List Window')
```

A.8.6 Process Manager

```
WindowVisibleSet('Process Manager')
```

A.8.7 Scripting Window

```
WindowVisibleSet('Scripting Window')
```

A.8.8 Spreadsheet

```
WindowVisibleSet('Spreadsheet')
```

A.8.9 Style Control

```
WindowVisibleSet('Style Control')
```

A.9 Help

A.9.1 About

```
About()
```

A.9.2 Dynamic Help

```
AppHelp('-profile ' + AppDocDir() + 'vida.adp')
```

A.9.3 Documentation

User Manual

```
AppOpenUrl('file:' + AppDocDir() + 'pdf/vida_manual.pdf')
```

Scripting Manual

```
AppOpenUrl('file:' + AppDocDir() + 'pdf/vida_python_api.pdf')
```

A.9.4 Online Support

FAQ

```
AppOpenUrl('http://www.eyesopen.com/support/faq/#vida')
```

User Manual (HTML)

```
AppOpenUrl('http://www.eyesopen.com/docs/html/vida/manual')
```

User Manual (PDF)

```
AppOpenUrl('http://www.eyesopen.com/docs/pdf/vida_manual.pdf')
```

Scripting Manual (HTML)

```
AppOpenUrl('http://www.eyesopen.com/docs/html/vida/api')
```

Scripting Manual (PDF)

```
AppOpenUrl('http://www.eyesopen.com/docs/pdf/vida_python_api.pdf')
```

Open License

```
AppOpenUrl('file:' + AppLicenseFile())
```

Open User Directory

```
AppOpenUrl('file:' + AppUserDir())
```

APPENDIX

B

Mouse Maps

B.1 VIDA Mouse Map

Table B.1: VIDA Mouse Map

Function	Mouse Buttons and/or Keyboard Modifiers
Zoom	Wheel
	XY motion while holding *Middle* button
	XY motion while holding *Left*+*Right* buttons
Rotate	XY motion while holding *Left* button
Rotate around Z axis	XY motion while holding *Left* button and *Alt*
Translate in XY plane	XY motion while holding *Left* button and *Shift*
Translate in Z plane	Wheel and *Alt*
	XY motion while holding *Left* button and *Alt*+*Shift*
Select	Click *Left* button
Select rectangle	XY motion while holding *Right* button
Select add	Click *Left* button while holding *Shift*
	Click *Left* button while holding *Control*
Select grow	Double click *Left* button
Select add grow	Double click *Left* button while holding *Shift*
	Double click *Left* button while holding *Control*
Select rectangle and add	XY motion while holding *Right* button and *Shift*
	XY motion while holding *Right* button and *Control*
Change clipping plane	Wheel and *Control*+*Shift*
Change far clipping plane	Wheel and *Alt*+*Control*
Change near clipping plane	Wheel and *Alt*+*Shift*
Text Scale	Wheel and *Control*
Menu	Click *Right* button
Adjust contour level	Wheel and *Shift*
Show labels	XY motion and *Control*

APPENDIX

C

Release Notes

C.1 Version 3.0.0

April 2007

Minor Bug Fixes:

1. Fixed a problem where on some graphics cards only aromatic rings would be drawn when a molecule was in stick mode

Major Bug Fixes:

1. Upgraded to OEChem 1.5 which fixed a number of bugs

Improvements:

1. Anti-aliasing of text and 2D depictions
2. Much better control over placement of dock windows
3. Dock windows can be stacked on top of each other in tabs
4. Single toolbar at the top of the application
5. Additional popup toolbars placed around edge of main window
6. Enabled grid contour transparency
7. Hydrogen style is now a molecule specific property instead of a global property
8. C-Alpha trace display is now a molecule specific property instead of a global property
9. Ribbon display is now a molecule specific property instead of a global property
10. Many miscellaneous 3D rendering improvements and accelerations

11. Many miscellaneous user interface improvements

New Features:

1. Added dynamic help system
2. Added new 2D depiction color schemes: black on white, white on black, color on white and color on black
3. Added new molecule coloring schemes: amino, chain, group, cpk, cpknew, shapely, bfactor, formal charge, and partial charge
4. Added new surface hydrophobicity scales: charifson, eisenberg, kyte-dolittle, and white-octanol
5. Added support for reading attached grids and surfaces in OEB files
6. Added support for reading Spicoli geometric annotations
7. Added support for reading FRED receptor files
8. Added new hydrogen style: polar hydrogens only
9. Added multiple new atom and bond label types (including one for OEChem generic data)
10. Added new 3D display mouse maps for Coot and PyMol

C.2 Version 2.1.2

July 2006

Minor Bug Fixes:

1. Fixed "MolecularEnergy()" calculation method in spreadsheet to correctly report energies on multi-conformer molecules
2. Fixed "SpreadsheetFilter()" command such that it operates on the specified spreadsheet instead of the current spreadsheet
3. Ensured that all current environment variables are properly passed to processes started from the process manager
4. Fixed the icon on the hydrogen display button to accurately reflect what will occur when the button is pressed
5. Fixed a Mac OS X bug where double-clicking on a molecule file started VIDA but did not actually load the file that was clicked on

Major Bug Fixes:

1. Upgraded to OEChem 1.4.1 which fixed a number of bugs associated with handling bad input files
2. Fixed a cross-platform compatibility issue in state files

Improvements:

1. Updated menu system to be compliant with standard UI design practices (such as '...' usage when menu option launches a dialog)
2. Updated menu system on Mac OS X to follow common Mac paradigms (e.g. "Open Recents" and "Preferences..." options)
3. Reorganized File menu for more logical placement of options
   (a) Collapsed "New > List >" submenu structure to a single "New List >" submenu which appears after the Open, Save block
   (b) Added "New Molecules >" submenu which contains a "From Split" submenu which provides the "Split Molecules" functionality from the old Edit menu. In VIDA 2.2 a "From Merge" will join this option
   (c) Renamed "Clear All" option to "Clear" and moved up in the menu above Save block
   (d) Removed "Abort Script" option as it was redundant (Stop button next to progress bar performs this function) and also could generally not be accessed while a script was running
4. Reorganized Edit menu for more logical placement of options as well as to provide some commonly performed operations in a single place
   (a) Moved "Split Molecules" functionality to File menu
   (b) Added new "Select" submenu which contains the following options: All Visible, Query, Invert, and None
   (c) Added new "Mark" submenu which contains the following options: All, Query, None
   (d) Added new "Hide" submenu which contains the following options: All, Selected, Marked, Not-Selected, Not-Marked, and None
   (e) Removed the following options as they were redundant given the above additions: Hide All, Clear Marked
5. Alphabetized list of available windows in the Window menu
6. Batch shell window no longer appears when running on Windows
7. Redesigned process manager window for easier use
8. Significantly improved the quality of POV-ray exports and reduced the output file size
9. Improved the rendering of electrostatic grids

New Features:

1. Added “Copy Image” function under the Edit menu which takes a screenshot of the current main window and places it on the clipboard
2. Added “Map Index” atom label corresponding to the value returned by the OEChem function OEAtomBase::GetMapIdx().

C.3 Version 2.1.1

March 2006

Minor Bug Fixes:

1. Fixed a problem where extra atom labels could appear on molecules when saved to state
2. Fixed a problem where CTRL-mouse over atom displays were not seen when using non-VIDA mouse maps
3. Fixed a bug in the “Text Scale” slider where its position did not always reflect the current scale being used
4. Fixed a bug where OE_DIR and OE_LICENSE environment variables were not being properly exported to the Process Manager on Mac OS X such that spawned applications could not find a valid oe_license.txt file
5. Fixed a bug where exporting a CSV file on Microsoft Windows attached an extra return character causing a blank line to appear between each row when read into Microsoft Excel
6. Fixed a few cases where the display of depictions in the spreadsheet was not obeying the set preferences
7. Fixed a bug where extra calls to undo cleared the redo stack
8. Turned off the writing of empty data fields when writing molecules to SD or OEB formats. This can be turned back on if truly desired from the Preferences

Major Bug Fixes:

1. Fixed a problem with reading spreadsheet information in state files generated by VIDA 2.0.x which caused extra tabs to appear in the spreadsheet
2. Fixed a problem with writing spreadsheet information correctly in VIDA 2.1
3. Fixed a crash when reading a badly formatted state file (potentially generated by the bug fixed above)
4. Fixed a crash when attempting to import a non-standard whitespace delimited spreadsheet (whitespace is not a supported spreadsheet delimiter)

5. Fixed a crash on certain video cards when attempting to display “fixed size text” labels
6. Fixed a crash caused by dragging state files onto the application icon in order to open them
7. Fixed an application hang when reading badly formatted XPLOR maps
8. Fixed an application hang when reading a badly formatted OEB file
9. Fixed a memory leak generated when running a large number of scripting commands

Improvements:

1. Filled out Sybyl mouse map in more detail
2. Added “fixed size text” option for labels (see Preferences) such that the label font size remains fixed independent of zooming in the scene, however the font size can be adjusted using the “Text Scale” slider in the Style Bar window
3. Added the ability to specify the font for labels when using “nice fonts” for labels (see Preferences)
4. Added a “custom” option to the atom and bond label dialogs to allow for easier specification of custom labels without having to use scripting commands
5. Improved the drawing of labels such that they always appear in front of the object they are labeling
6. Improved the ability to see the selected state of partially selected residues in the sequence view
7. Added file dialog filter option to allow reading of ASCII formatted grids (*.agd)
8. Added support for native builds on Intel Macs

C.4 Version 2.1.0

January 2006

Minor Bug Fixes:

1. Fixed a number of preferences that were not being properly saved or obeyed by the application
2. Fixed the undo stack such the initial setup of the 3D display cannot be undone by accident
3. Fixed the display of angle monitor measurements from radians to degrees
4. Fixed the display of the text scale widget such that it is not hidden on low resolution monitors
5. Made sure that when saving images and state files, a proper file extension is added to the filename, even if one was not specified

Major Bug Fixes:

1. Fixed a crash when trying to generate electrostatic contours on extremely small molecules (e.g. water)

Features:

1. Added interactive text annotation of molecules
2. Added the ability to start, monitor, terminate, and load results from external processes
3. Added a protein sequence viewer as part of the 2D depiction widget
4. Added the `oechemlite` Python module, enabling significantly greater scripting functionality within VIDA
5. Added the ability to create arbitrary sphere monitors via the scripting API (see Python API for documentation of the new method `CreateSphereMonitor(...)`)
6. Added support for fullscreen display
7. Added new *cloud* visualization style for grids
8. Added support for copy and paste of molecules from ISIS/Draw
9. Added support for use of the Maestro mouse map in the 3D display
10. Added support for saving screenshots in JPG format in addition to PNG

Improvements:

1. Cleaned up the appearance of the right-click popup menus
2. Added a “Style” option to the right-click popup menus, when appropriate to the context, which mirrors the main “Style” menu
3. Added significantly greater detail to info labels created when holding down the “ctrl” (or “alt” on Mac) key when moving the mouse in the 3D or 2D views
4. Added two default filtered spreadsheets to the application: “Molecules” and “Proteins”
5. Filled out the scripting API for greater control of the GUI - in particular for menus and toolbar
6. Greatly improved the electrostatic coloring scheme for protein surfaces
7. Added the ability to change the font used by the application

C.5 Version 2.0.2

December 2005

Minor Bug Fixes:

1. Fixed a bug in the detection of a user's home directory in networked environments on Microsoft Windows
2. Fixed a bug in "measurement" mode where the first atom or bond picked was not being drawn as selected
3. Fixed a bug where spreadsheet filtering performed via the scripting interface would not behave as expected if the specified spreadsheet was not the same as the current spreadsheet
4. Fixed an uncommon situation in which the locked and visible controls in the list window do not work as expected
5. Fixed the handling of certain "escape characters" in filenames specified on the commandline, files dragged onto the VIDA icon, and the "startup.py" file such that they no longer prevent the opening of those files

Major Bug Fixes:

1. Fixed right-click on empty 2D depiction crash
2. Fixed spreadsheet import failure on linux
3. Fixed mainwindow snapshots such that file dialog does not obscure the image to be captured
4. Fixed license detection routines such VIDA will properly detect installed licenses and not cause start-up problems if the license file is moved
5. Fixed all other reported crashes not otherwise listed

   Improvements and Optimizations:

6. Added "Style" menu to the main menubar which replicates the functionality of the style control (useful in low-resolution settings)
7. Added "Center" and "Color" options to the popup dialogs
8. Enabled display of named network drives in the file dialog
9. Enabled use of native file dialogs instead of custom VIDA file dialog as a preference.
10. Improved 3D visualization support for lower-end graphics cards
11. Increased default line width in 3D scene for better visibility
12. Enabled transparent display of grids (via the "Style" menu)

13. Added simple spreadsheet import preview dialog to help select the proper column delimiter (tab vs. comma)
14. Added SpreadsheetCurrentSet() function which allows specification of current spreadsheet via scripting interface
15. Improved the SD file reader for better handling of badly formatted SD files
16. Improved the handling of situations in which the atom indices specified in input files differ from the order in which they are specified
17. Improved start-up time as license detection mechanisms have been significantly improved

Changes:

The API for the "Open" Python scripting command has changed. Previously, the "Open" command returned the ID of the first item in the file that was opened. Unfortunately, this proved to be a significant hurdle in the use of this function in Python scripts. As such, the "Open" command now returns a list of IDs corresponding to each item loaded from the file. Any VIDA scripts that depend on the return value of the 'Open" command will need to be changed from this:

```
id = Open("example.oeb")
DoSomething(id)
```

to:

```
ids = Open("example.oeb")
if ids:
  DoSomething(ids[0])
```

We have no plans to make any further API changes that would break current VIDA scripts. In fact, we debated for a long time whether to even make this change. However, this was a unique situation in which the current API just did not do what was needed in order to make good use of the scripting languge. So, instead of building a work-around, we all agreed that this would be the most appropriate solution.

BIBLIOGRAPHY

[1] Robert J. Fletterick, Raymond Matela, "Color-coded α-carbon models of proteins", *Biopolymers*, Vol. 21, pp. 999-1003, 1982.

[2] Matthew D. Eldridge, Christopher W. Murray, Timothy R. Auton, Gaia V. Paolini and Roger P. Mee, "Empirical scoring functions: I. The development of a fast empirical scoring function to estimate the binding affinity of ligands in receptor complexes", *Journal of Computer-Aided Molecular Design*, Vol. 11, pp. 425-445, 1997.

[3] Araz Jakalian, Bruce L. Bush, David B. Jack and Christopher I. Bayly, "Fast, Efficient Generation of High-Quality Atomic Charges. AM1-BCC Model: I: Method", *Journal of Computational Chemistry*, Vol. 21, pp. 132-146, 2000.

[4] Araz Jakalian, David B. Jack and Christopher I. Bayly, "Fast, Efficient Generation of High-Quality Atomic Charges. AM1-BCC Model: II: Parameterization and Validation", *Journal of Computational Chemistry*, Vol. 23, pp. 1623-1641, 2002.